THEORETICAL FUNDAMENTALS OF REAL-TIME VIRTUALIZATION FROM THE RESOURCE MANAGEMENT PERSPECTIVE

A Dissertation Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

By

Yu Li

December 2016

THEORETICAL FUNDAMENTALS OF REAL-TIME VIRTUALIZATION FROM THE RESOURCE MANAGEMENT PERSPECTIVE

Yu Li

APPROVED:

Albert M. K. Cheng, Chairman
Dept. of Computer Science

Kam-Hoi Cheng
Dept. of Computer Science

Weidong (Larry) Shi
Dept. of Computer Science

Omprakash Gnawali
Dept. of Computer Science

Yuhua Chen
Dept. of Electrical & Computer Engineering

Dean, College of Natural Sciences and Mathematics

THEORETICAL FUNDAMENTALS OF REAL-TIME VIRTUALIZATION FROM THE RESOURCE MANAGEMENT PERSPECTIVE

An Abstract of a Dissertation

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

By

Yu Li

December 2016

Abstract

A Virtual Machine Monitor (VMM) partitions a host physical machine into a group of Virtual Machines (VMs). Typically, a VM machine only preempts a part of a dedicated physical resource temporally or spatially. This fact greatly impacts the real-time task scheduling in VMs because most traditional real-time scheduling theories are based on dedicated resources. The real-time community has introduced some Hierarchical Real-Time Scheduling Models to address this issue. Among them, the Regularity-based Resource (RRP) Model is able to provide maximal transparency for task scheduling. However, current theoretical results on the RRP Model are still far from the complete theoretical fundamentals required by a real-time VMM. At the resource level, only a naive algorithm has been found for resource partitioning. At the task level, only the Periodic Task Model is investigated, and even for this task model, only one simple case has been considered. This work explores the RRP Model at both the resource and task levels. On the one hand, it is the first to solve the resource partitioning problem with both global and partitioned strategies. On the other hand, it solves the task scheduling problem with a strong result that the classic task scheduling problem in the RRP Model can be easily transformed into an equivalent problem on a dedicated resource. With these theory enhancements, a 2-layer real-time resource model is presented and the theoretical fundamentals of a real-time VMM are fully established from resource management perspective.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The Virtual Machine (VM) technology has been developed in the recent decade. Some well-known Virtual Machine Monitors (VMM), such as Virtual PC [37], Virtual Box [38], Xen [39], Vmware [40], and KVM [41], have reached the practical stage and been widely used in industry. Resource management is a fundamental problem in a VMM. To support multiple VMs on a single physical platform, a hierarchical resource model is required. Figure 1.1 shows a two-layer resource structure, where a physical resource (like a CPU) is divided into a group of virtual resources by some resource-level scheduling algorithms, and a virtual resource (like a virtual CPU) is assigned to a VM. Ideally, each VM acts as a black box and runs its individual task-level scheduler. There are several resource-level schedulers [42, 43, 44] providing good performance for general-purpose VMs in different application scenarios.

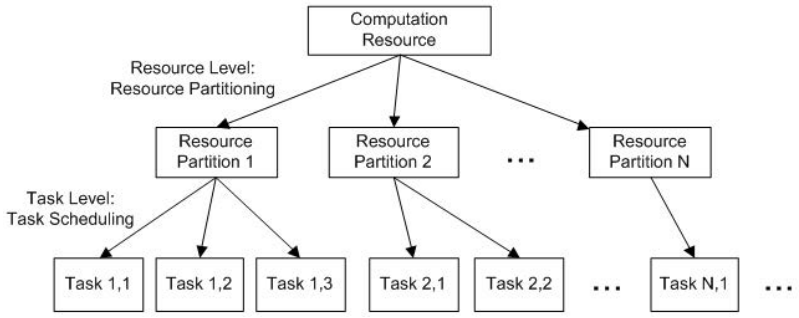


Figure 1.1: A 2-layer Hierarchical Real-time System

However, there is a different story in real-time systems, which are implemented with embedded microprocessors and associated hardware resources which constitute over 90% of the computing systems in use. A principle of a real-time system is precise-timing control. Basically, each task in such a system has a soft or hard deadline to make sure that it can finish in a "real-time" manner. With such a constraint, researchers have developed a collection of scheduling policies. Earliest Deadline First (EDF) and Rate Monotonic (RM) [8] are two of the most famous. A foundational work of the real-time community is to investigate the applicability of these policies in various scenarios. Typically, several common task models are defined first as follows:

• The **periodic** task model - the requests of each task arrive regularly; two successive requests are separated by exactly the same time delay, called the *period*.

• The **sporadic** task model - the requests of each task arrive irregularly with each arrival separated from the preceding one by at least a certain time delay called the *minimum-separation time*.

• The **aperiodic** task model - the requests of an aperiodic task arrive irregularly with no minimum-separation time.

For the combination of a scheduling policy and a task model, researchers try to find the sufficient and necessary *schedulability test** that ensures a given task set can meet the real-time constraint. Sometimes for convenience, people also would like to obtain a *utilization bound* equation for such a combination. A task set is guaranteed to be schedulable if its utilization is lower than this bound.

However, these useful scheduling techniques could not be applied in a real-time virtual machine because a virtual machine runs on a virtual resource instead of a dedicated physical resource because a virtual resource only uses only a fraction of the time on a physical resource. Researchers and developers have to reinvestigate each scheduling problem with huge efforts, which include obtaining the new schedulability tests and utilization bounds to adapt to the virtulization scenario. There are several Hierarchical Real-Time Scheduling (HiRTS) models for the purpose of real-time virtulization, such as the Periodic Model [12], the EDP Model [14], the Bounded-delay Model [2], and the Regularity-based Model [1]. Table 1.1 shows some of their features. We focus on two important factors of these models: the manner of Resource Partitioning (virtual resource scheduling) and the Changes of Task Scheduling.

This table shows that there is some trade-off between these two factors. The Periodic and EDP Models schedule virtual resources more easily but need to make heavier changes for task scheduling. On the other hand, the Bounded-Delay and

*Due to the difficulty, only the necessary schedulability tests have been obtained in some scenarios.

Regularity-based Models are the exact opposites. The next question is: When we choose the resource model of a real-time VMM, which factor is more important between these two? If we stand with the real-time application developers, we shall definitely choose the hinder. Light changes of task scheduling mean less effort to modify the real-time applications when the developers migrate them from a physical machine to a virtual machine. We give an example to illustrate our opinion. The Regularity-based Model [7] could keep the utilization bound of the EDF policy unchanged for some scenarios with the Periodic Task Model. Suppose there is a real-time application, GEEK, holding these assumptions (using EDF and the Periodic Task Model). GEEK is now running on a 1-GHz CPU core. We want to port it to a 2-GHz CPU core using the real-time virtualization technology. The problem is that we have limited details about the real-time tasks in GEEK. If the utilization bound on the virtual CPU need to be recomputed, we cannot obtain the exact size of the required virtual CPU to run GEEK. Even though we can take an estimate and use a larger value, there is still no full real-time guarantee. Things become easy if the utilization bound is unchanged; we only need to assign its weight as 0.5.

Therefore, due to the importance of Factor II in Table 1.1, we choose the Regularity-based Model as the foundation of this dissertation. Our work is the extension of [7]

	Factor I: Difficulty on Resource Partitioning	Factor II: Change on Task Scheduling
Periodic / EDP	easy	heavy
Bounded-delay	no effective algorithm found	moderate
Regularity-based	a naive algorithm found	light

Table 1.1: Several HiRTS Models

and [1]. Although they have presented some elementary results of the Regularity-based Model, they are still far from the complete theoretical fundamentals needed to build a real-time VMM. First, only a naive algorithm has been found for resource partitioning. Second, only the Periodic Task Model is investigated for the task scheduling, and even for this task model, only one simple case is investigated.

1.2 Problems and Challenges

Whereas traditional strategies allocate a dedicated computation resource to each real-time application with multiple tasks, HiRTS integrates a group of multi-task applications on a single computation resource. In this approach, the Resource Partition [2] acts as an intermediate layer as shown in Figure 1.1. Each resource partition uses only a fraction of the time on the computation resource and a multi-task application is assigned to it in a one-on-one manner. Typically, each HiRTS model has its own definition of resource partition containing some parameters. Based on this definition, researchers investigate how to divide the computation resource into resource partitions at the resource level and schedule real-time tasks on a resource partition at the task level. We call them the *resource partitioning* problem and the *task scheduling* problem, respectively.

In our system, a computation resource could be a single resource or an identical multiresource. A time unit is a system-defined unit of time for the purpose of scheduling, which means that there is no (either partition or task) preemption inside a time unit. Therefore, we may assume that all time parameters in the system are

integers without loss of generality. We use a non-negative integer, t, to tag a time unit released at time, t. When the computation resource is an identical multiresource, there are multiple time units t in the system. A resource partition only preempts a fraction of the time, and it cannot contain two time units released at the same time. Since a resource partition is not always available at all times, the existing scheduling techniques, like schedulability tests and utilization bounds, might not work without changes. We formally define the transparency of resource partitions as follows:

*A resource partition is **transparent** for scheduling policies and task models if we can directly use the corresponding single-resource scheduling techniques to solve its task scheduling problem.*

Let's address the problems we shall discuss in the dissertation. At the resource level, a basic problem is how to schedule resource partitions on a single resource. We use (P1) Res-Single to represent this problem. In the multiresource scenario, there are two dominating categories of scheduling algorithms: global scheduling and partitioned scheduling. Global scheduling allows resource partitions to migrate between different computation resource units, while partitioned scheduling does not. They may dominate each other depending on different characteristics of computation resources. Generally, if the migration overhead is relatively small, global scheduling could provide higher efficiency; Otherwise, partitioned scheduling could perform better. Therefore, when considering the resource partitioning problem, we need to investigate both global scheduling and partitioned scheduling. We call them (P2) Res-Global and (P3) Res-Partitioned problems, respectively.

For task scheduling, we shall discuss the schedulability problems for several popular task models. One task model we consider is the Periodic Task Model, in which two successive requests of each task are separated by exactly the same time interval, called its period. In this model, a task t_i is denoted by (c_i, p_i, d_i, o_i), where c_i, p_i, d_i and o_i are its execution time, period, deadline, and offset, respectively. Most current HiRTS Models have investigated a simple case of the Periodic Task Model, in which each task's deadline is the same as its period and its offset is 0. We call (P4) Task-Periodic-Simple as the schedulability problem for this simple Periodic Task Model, and (P5) Task-Periodic-Generic as the one for the general Periodic Task Model. Another considered task model is the Sporadic Task Model, in which two successive requests of a task are separated by at least a time interval, called its minimum separation time. Similarly, we use (P6) Task-Sporadic to represent the schedulability problem for the Sporadic Task Model.

There are several HiRTS resource models, such as the Regularity-based (RRP) Model [1, 3], the Bounded-Delay Model [2, 3], the Periodic Model [12], and the EDP Model [14]. The Periodic Model (or Constant Bandwidth Server [36]) is the most popular one due to its simplicity for resource partitioning. Since a resource partition in this model is defined similarly to a periodic real-time task, the existing periodic task-scheduling techniques can be used to solve its resource-partitioning problem without changes. However, schedulability tests have been found only in the simplest case (P4) Task-Periodic-Simple for task scheduling in this model. Due to the blacked-out intervals without any computation time available, researchers have been stuck in more complicated problems, such as (P5) Task-Periodic-Generic and

(P6) Task-Sporadic.

Problem	the Periodic Model	the RRP Model
(P1) Res-Single	Converted to a Periodic Task Scheduling Problem	Primitive Results [1, 3]
(P2) Res-Global	Same As Above	Primitive Results [4]
(P3) Res-Partitioned	Same As Above	No Result
(P4) Task-Periodic-Simple	New Schedulability Tests for Periodic Partitions [12]	New Schedulability Tests for Regular and Irregular Partitions [1, 3]
(P5) Task-Periodic-Generic	No General Result	No General Result
(P6) Task-Sporadic	No General Result	No General Result

Table 1.2: The Current State of the Art: the Periodic Model and the Regularity-based Model

Besides the Periodic Model, other resource models have not received sufficient attention because they lacked effective resource partitioning algorithms. Recently, the RRP Model has drawn attention. Other than the Periodic Model, this model tends to evenly distribute the time units on a resource partition, and a parameter 'regularity' is used to restrain the time-unit distribution. A resource partition is called a Regular Partition when its regularity is minimal, which causes its time-unit distribution to be almost even. This idea was originally introduced in [7], and developed in [1, 3] for the single-resource scenario with some primitive results. Li and Cheng [4] first applied it onto a multiresource platform with global scheduling.

In Table 1.2, we summarize the current state of the art of the Regularity-based

and the Periodic Models. As mentioned, We chose the RRP Model as the foundation of our work because it is the only one with the potential for transparent real-time virtualization, but there are still some challenges to realize it.

The major challenge is to alleviate the RRP Model's weaknesses listed in Table 1.2. First, the current results on (P1) Res-Single and (P2) Res-Global are very primitive. Since the scheduling problem of regular partitions is announced NP-hard, the current solutions fall into the category of approximation algorithms. Mok and Feng [1, 3] gave an initial solution to schedule regular partitions on a single resource, where the weight of each regular partition is approximated by the values in an infinite sequence $\langle 1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, ... \rangle$. Li, Cheng, and Mok [4] extended this algorithm to the uniform multiresource scenario with a global-scheduling strategy. However, they did not consider other approximation sequences, and their results have not been proven optimal. We shall consider comprehensive cases of approximation sequences to improve the overall resource utilization and scheduling performance. Second, there has been no solution for (P3) Res-Partitioned in the RRP Model. On most types of multiresources, the migration overhead due to global scheduling is significant [33, 34]. Most practical scheduling algorithms in the multiresource scenario use a partitioned scheduling strategy. Therefore, it is important for us to invent a new solution for (P3) Res-Partitioned in the RRP Model. Last but not least, only (P4) Task-Periodic-Simple has been considered for task scheduling in the RRP Model. It is not convincing to show the transparency property on task scheduling based on such a simple result. Hence, we shall first solve (P5) Task-Periodic-Generic and (P6) Task-Sporadic for the RRP Model.

1.3 Other Related Work

Besides the mentioned HiRTS resource models, there exists several other theoretical works of hierarchical scheduling for uniresource platforms [45, 46, 47] and for multiresource platforms [11, 48, 49, 50, 51, 52]. However, none of them can achieve transparent real-time virtulization. From a practical perspective, there exist a large body of practical frameworks on hierarchical real-time scheduling. Initially, they were implemented within OS kernels [53, 54, 58, 59, 60], where thread groups were used for precise timing control. Recently, most efforts were transferred to Virtual Machine hypervisors, including [55] within the L4/Fiasco hypervisor, [56] within the KVM hypervisor and [57, 61, 62, 63, 64, 65] within the Xen hypervisor. However, all of these works were implemented based on servers (virtual resources) with or without fixed periods. They basically apply scheduling techniques similar to real-time task scheduling. Priority and Running Queues are widely used in their implementations. Therefore, none of them has achieved transparent real-time virtualization. On these platforms, people have to apply tedious changes on the existing task-level schedulers.

1.4 Organization

The rest of this thesis is organized as follows. We overview the Regularity-based Resource Model in Chapter 2. Then, we present our global and partitioned resource-partitioning algorithms in Chapter 3 and 4, respectively. In Chapter 5, we show our transparent task-scheduling techniques on regular partitions. Finally, we draw the

conclusion in Chapter 6. The content of Chapter 3 and 5 has been published in [13] and [35], respectively.

Chapter 2

Background: The Regularity-based Resource Model

2.1 Resource Partitions in the RRP Model

In this chapter, we introduce some basic definitions and results of the Regularity-based Resource (RRP) Model. In this model, the only parameter of a resource partition is its weight – a rational number between 0 and 1 showing the fraction of its utilization, and time units on the computation resource are periodically assigned to it. As shown in Def. 2.1, a partition assignment describes how such a resource assignment happens, which is specified by a period, p, and a sequence, T, containing q time units in the first period. To avoid ambiguities, we assume p, q are co-prime.

Definition 2.1 *A **partition assignment** $\mathcal{A}$ is a tuple (T, p), where $T = \langle t_0, t_1, ..., t_{q-1}$:*

$0 \leq t_0 < t_1 < ... < t_{q-1} < p\rangle$ *is the time-unit sequence;* p *is the period and* p, q *are co-prime.*

The *supply function* $S_{\mathcal{A}}(t)$ of a partition assignment $\mathcal{A}$ indicates the total number of time units that are available in $\mathcal{A}$ from time 0 to t, which is recursively described in Def. 2.2. For ease of discussion, we announce a naming convention upon a sequence T that $|T|$ denotes its element number (could be infinite) and $T|_i$ denotes its i-th element.

Definition 2.2 *Given a partition assignment* $\mathcal{A} = (T, p)$ *where* $|T| = q$*, its supply function* $S_{\mathcal{A}}(t) =$

$$\begin{cases} 0 & if\ t = 0; \\ S_{\mathcal{A}}(t-1) & if\ 0 < t < p\ and\ t-1 \notin T; \\ S_{\mathcal{A}}(t-1) + 1 & if\ 0 < t < p\ and\ t-1 \in T; \\ q \cdot \lfloor \frac{t}{p} \rfloor + S_{\mathcal{A}}(t\ mod\ p) & otherwise, t \geq p. \end{cases}$$

At each time instant, the *instant regularity* indicates the difference between the actual and ideal resource allocation on a partition assignment.

Definition 2.3 $I_{\mathcal{A}}(t)$ *denotes the instant regularity of* $\mathcal{A}$ *at time* t *where* $I_{\mathcal{A}}(t) = S_{\mathcal{A}}(t) - t \cdot \frac{q}{p}$.

Figure 2.1 shows an example of a partition assignment of weight $\frac{4}{7}$. The polyline represents the supply function $S_{\mathcal{A}}(t)$ showing the actual resource allocation to $\mathcal{A}$. The ideal allocation line has a slope of $\frac{4}{7}$ equal to the partition weight, indicating the ideal resource allocation to $\mathcal{A}$. The instant regularity of $\mathcal{A}$ at time 2 is $I_{\mathcal{A}}(2) =$

$S_{\mathcal{A}}(2) - 2 \cdot \frac{4}{7} = \frac{6}{7}$.

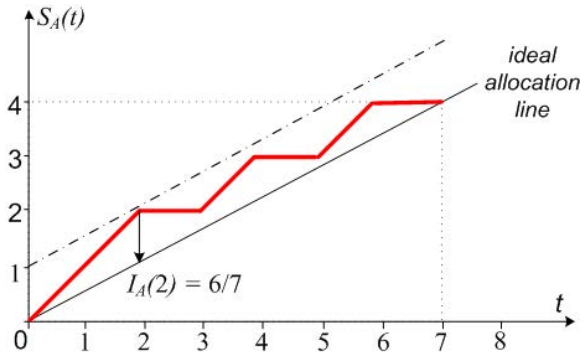


Figure 2.1: A Partition Assignment of Weight $\frac{4}{7}$

A *regular-partition assignment* minimizes the deviation range of its instant regularity (see Def. 2.3). The size of this range is limited to less than 1 as shown in Figure 2.1. We would like to mention that this bounded-deviation-range strategy is also used in the P-fair algorithm [10, 11], but the range in P-fair is bounded by 2. Another difference is that P-fair puts this restriction on tasks and only considers a single-layer scheduling problem. To achieve the transparency property in HiRTS, our work puts this bounded-deviation-range restriction on resource partitions, on which we consider the task-scheduling problem further.

Definition 2.4 *A partition assignment* $\mathcal{A}$ *is* ***regular*** *if and only if* $\forall t_1, t_2, |I_{\mathcal{A}}(t_1) - I_{\mathcal{A}}(t_2)| < 1$.

We can also easily observe the different changes on the instant regularity when a time unit is allocated or not, as the following lemma shows.

Lemma 2.1 *Suppose* $\mathcal{A} = (T, p)$ *is a regular-partition assignment and* $|T| = q$*, then*

$\forall t \in [0, p)$,

$$I_{\mathcal{A}}(t+1) - I_{\mathcal{A}}(t) = \begin{cases} 1 - \frac{q}{p} & if\ t \in T; \\ -\frac{q}{p} & otherwise. \end{cases}$$

Proof. Immediately follows from Def.s 2.2 and 2.2. □

Let $\beta_{\mathcal{A}}$ denote $\mathcal{A}$'s minimum instant regularity. Since $I_{\mathcal{A}}(0) = 0$, by Def. 2.4, $\beta_{\mathcal{A}} \in (-1, 0]$. By Lemma 2.1, $\beta_{\mathcal{A}}$ has p possible values: $0, \frac{1}{p}, ..., \frac{p-1}{p}$, and there are p different regular-partition assignments, respectively.

Definition 2.5 *Suppose p, q are co-prime and $d \in [0, p)$, $\mathcal{A}_d^{(p,q)}$ denotes the regular-partition assignment whose weight is $\frac{q}{p}$ and minimum instant regularity is $-\frac{d}{p}$.*

Next, we show how to compute the time units of a regular-partition assignment when its weight and minimum instant regularity are known.

Lemma 2.2 $\mathcal{A}_d^{(p,q)} = (\langle \lfloor \frac{i \cdot p + d}{q} \rfloor : i = 0, 1, ..., q-1 \rangle, p)$.

Proof. By Def.s 2.4 and 2.5, $\forall t, I_{\mathcal{A}}(t) \in [-\frac{d}{p}, 1 - \frac{d}{p})$. Suppose t_0 is the first time unit of $\mathcal{A}_d^{(p,q)}$, then by Lemma 2.1, $I_{\mathcal{A}}(t_0) = -t_0 \cdot \frac{q}{p}$ and $I_{\mathcal{A}}(t_0 + 1) = 1 - (t_0 + 1) \cdot \frac{q}{p}$. It follows that $-t_0 \cdot \frac{q}{p} \geq -\frac{d}{p}$ and $1 - (t_0 + 1) \cdot \frac{q}{p} < 1 - \frac{d}{p} \Rightarrow \frac{d}{q} - 1 < t_0 \leq \frac{d}{q} \Rightarrow t_0 = \lfloor \frac{d}{q} \rfloor$. Similarly, the second time unit is $\lfloor \frac{p+d}{q} \rfloor$; ...; the q-th time unit is $\lfloor \frac{(q-1)p+d}{q} \rfloor$. □

Now is a good time to explain why we only need to consider the case in which p, q are co-prime, where p is the period of a regular-partition assignment and q is the number of time units in each period. Let us consider a regular-partition assignment $\mathcal{A}$ whose period is $n \cdot p$ and there are $n \cdot q$ time units in each period where p, q are

co-prime. Following similar calculations in Lemmas 2.1 and 2.2, $\beta_{\mathcal{A}}$ has p possible values and there are only p (not $n \cdot p$) different time unit sequences for $\mathcal{A}$. Actually, we could also find that $\mathcal{A}$ is just an n-time repetition of a regular-partition assignment whose period is p and weight is also $\frac{q}{p}$. Therefore, the co-prime case is able to cover all other cases.

We are ready to define regular partition. A regular partition represents a fraction of the computation resource and it only admits regular-partition assignments.

Definition 2.6 *Given co-prime p, q, a* ***regular partition*** *of weight $\frac{q}{p}$ periodically preempts q time units within each period, p, and its partition assignment is regular.*

A regular partition of weight $\frac{q}{p}$ has p different possible assignments when p, q are co-prime. When scheduling a task set on this regular partition, we do not know exactly which partition assignment it has because this is a detail of resource partitioning at the resource level shown in Figure 1.1. In some scenarios, the assignment to a regular partition could even be changed. To deal with this situation, a task set is claimed to be schedulable on a regular partition if and only if it is schedulable on each possible assignment of this regular partition. For convenience, we define $T_0(p, q)$ and $T(p, q, \delta)$ as follows:

Definition 2.7 $T_0(p, q) = \mathcal{A}_0^{(p,q)}$ *is called a* ***standard regular-partition assignment****, whose minimum instant regularity is 0.*

For example, Figure 2.1 shows $T_0(7, 4) = \langle 0, 1, 3, 5 \rangle$. The motivation to point out such a special regular-partition assignment is that any other regular-partition

assignment with the same weight can be easily obtained from it by applying a right-shift operation. Therefore, we usually only need to check the properties of regular-partition assignment on the standard ones. Figure 2.2 shows an example containing such kind of right shifts.

Definition 2.8 *$T(p, q, \delta)$ represents the time-unit sequence right shifted from $T_0(p, q)$ by δ time units, where a modulus operation is applied when a time unit is out of $[0, p)$. Specially, $T(p, q, 0) = T_0(p, q)$.*

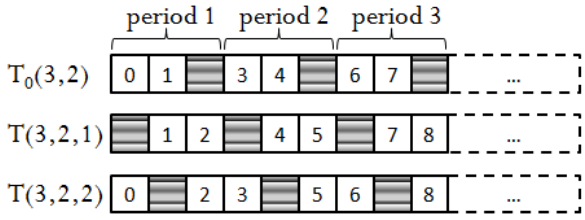


Figure 2.2: Right-Shifting Regular Partition $(T_0(3, 2), 3)$

2.2 Containment Relation between Regular Partitions

The following discussion answers the question of whether a regular-partition assignment can be a part of another one in some special cases. These results are very important for our approximation method of regular-partition scheduling. We always use it to determine counter examples when we check the feasibility of an approximation sequence. We formally define a partial order $\prec_{rp}$ to indicate the containment

relation between regular partitions.

Definition 2.9 $\frac{q_1}{p_1} \prec_{rp} \frac{q_2}{p_2}$ *if and only if* $\exists\delta$, $T(p, \frac{q_1}{p_1} \cdot p, \delta) \subset T_0(p, \frac{q_2}{p_2} \cdot p)$, *where* $p = LCM(p_1, p_2)$.

For convenience, we use RP_w to represent a regular partition whose weight is w. Then, we can also describe $\prec_{rp}$ like this: $w_1 \prec_{rp} w_2$ if and only if RP_{w_1} could be contained by RP_{w_2}. This containment relation is checked frequently while scheduling regular partitions. The following two lemmas investigate this containment relation in some special cases.

Lemma 2.3 *If* $w_1 \prec_{rp} w_2$ *where* $w_1 + w_2 < 1$ *and* $w_2 < 2w_1$, *then* $\exists n > 0$, $w_1 = \frac{n+1}{4n+3}$, $w_2 = \frac{2n+1}{4n+3}$.

Proof. We only need to show that if $T(p, r, \delta) \subset T_0(p, q)$ where $q + r < p$ and $r < q < 2r$, then $\exists n > 0$, $\frac{q}{p} = \frac{2n+1}{4n+3}$; $\frac{r}{p} = \frac{n+1}{4n+3}$.

Let S (resp. S') denote the infinite sequence including all time units on the regular partition $T_0(p, q)$ (resp. $T(p, r, \delta)$), then S' is a subsequence of S. Let $d = \lfloor \frac{p}{q} \rfloor$ and $d' = \lfloor \frac{p}{r} \rfloor$. By Lemma A1, any span size in S (resp. S') is either d or $d + 1$ (resp. d' or $d' + 1$). Meanwhile, since $q + r < p$ and $r < q < 2r$, we have $d \leq d' \leq 2d + 1$.

CASE 1: $d \geq 3$. Since $d \leq d' \leq 2d+1$ and $2d > (d+1)+1$, the size of any span in S' can only be chosen from either $\{d, d+1\}$ or $\{2d, 2d+1\}$. The first case is impossible because there must be a pair of neighboring time units in S' separated by a time unit in $S - S'$, the distance of which is not less than $2d$. In the second case, the time units in S must be assigned to S' and $S - S'$ alternately. It follows $q = 2r$, which

contradicts $q < 2r$.

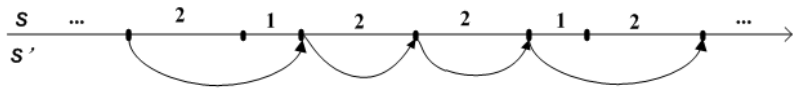
CASE 2: $d = 1$. Then $\left\lfloor \frac{p}{q} \right\rfloor = d = 1; p > q + r; r < q < 2r$

$\Rightarrow \frac{1}{2}p < q < \frac{2}{3}p$; $\frac{1}{4}p < \frac{1}{2}q < r < 1 - q < \frac{1}{2}p$

$\Rightarrow \frac{1}{2} < \frac{q}{p} < \frac{2}{3}$; $2 < \frac{p}{r} < 4$.

From $\frac{1}{2} < \frac{q}{p} < \frac{2}{3}$, there are no neighboring 1-size spans in S *; from $2 < \frac{p}{r} < 4$, $d' = \lfloor \frac{p}{r} \rfloor = 2$ or 3.

CASE 2.1: $d' = 2$. Suppose there are x (resp. y) times 1-size (resp. 2-size) spans in the first period of S. Then $x + 2y = p, x + y = q \Rightarrow y = p - q$.



As shown in the figure, since there are no neighboring 1-size spans in S, each 2-distance span in S' corresponds to a 2-distance one in S, and each 3-distance span in S' corresponds to a 1-distance one and a 2-distance one in S. Suppose there are x' (resp. y') times 3-size (resp. 2-size) spans in the first period of S', then $x' = x, y' = y - x \Rightarrow r = x' + y' = y \Rightarrow r = p - q$, which contradicts $q + r < p$.

CASE 2.2: If $d' = 3$, any span size in S' is 3 or 4, then each span in S' corresponds to at least two consecutive spans (size 1 or 2) in S. It follows $q \geq 2r$, which contradicts $q < 2r$.

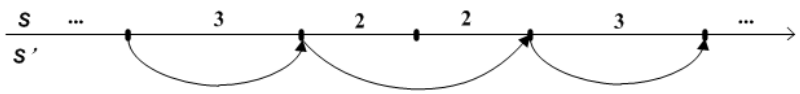
CASE 3: $d = 2$. Then $2 \leq d' \leq 5$.

*Three consecutive preemptive time units increase the instant regularity on S by $3 \cdot (1 - \frac{q}{p}) > 1$, which contradicts Def. 2.4.

CASE 3.1: $d' = 2$ is impossible because there must be a pair of neighboring time units in S' separated by a time unit in $S - S'$, the distance of which is not less than $2d = 4$.

CASE 3.2: If $d' = 4$ or 5, the time units in S must be assigned to S' and $S - S'$ alternately. It follows $q = 2r$, which contradicts $q < 2r$.

CASE 3.3: If $d' = 3$, the span size in S' is 3 or 4. As shown in the figure, each 3-size (resp. 4-size) span in S' corresponds to a 3-size span (resp. two consecutive 2-distance spans) in S. Also, 4-distance spans must exist in S'; otherwise, $r = q$, which contradicts $r < q$.



Suppose there is a series of consecutive 2-size spans between two nearest 3-size spans in S, whose number is $2n > 0$, then $\frac{p}{q} \in (2 + \frac{1}{2n+2}, 2 + \frac{1}{2n})$ [†]. Thus, the number of consecutive 2-size spans between any two nearest 3-size spans in S equals $2n$. It follows that $\frac{q}{p} = \frac{2n+1}{4n+3}$ and $\frac{r}{p} = \frac{n+1}{4n+3}$. □

Lemma 2.4 *If* $w_2 \leq \frac{1}{3}$ *and* $\frac{w_2}{2} < w_1 < w_2$, $w_1 \not\prec_{rp} w_2$.

Proof. Immediately follows from Lemma 2.2. □

[†]With the following preemptive time unit, 2n times consecutive 2-size spans increase the instant regularity by $2n+1-(4n+1)\frac{q}{p} < 1$ (Def. 2.3) $\Rightarrow \frac{p}{q} < 2+\frac{1}{2n}$. Similarly, excluding the first preemptive time unit, two times 3-size spans and 2n times 2-size spans decrease the instant regularity by $(4n+5)\frac{q}{p} - (2n+1) < 1 \Rightarrow \frac{p}{q} > 2 + \frac{1}{2n+2}$.

For example, by Lemma 2.4, we easily conclude that a $\frac{1}{3}$-weight regular-partition assignment does not contain any $\frac{1}{4}$-weight one; by Lemma 2.3, a $\frac{5}{9}$-weight regular-partition assignment does not contain any $\frac{1}{3}$-weight ones.

Chapter 3

Magic7: Optimal Global Resource Partitioning

[†]With the rapid development of both software and hardware, large-scale, open real-time systems have become increasingly popular. To improve resource utilization, these systems intend to deploy a group of real-time applications on one physical platform. Instead of running on a dedicated physical resource, the real-time tasks belonging to one application run on a real-time virtual resource [2], which can only occupy a temporal partition of the physical resource. Thus, a new type of real-time system called the hierarchical real-time system emerges.

A resource partition [2] describes how to assign the time intervals on a physical

[†]The content of this chapter has been published. **©2012 IEEE.** Reprinted, with permission, from Yu Li, Albert M. K. Cheng, Static Approximation Algorithms for Regularity-based Resource Partitioning, Real-Time Systems Symposium, December 2012.

resource (e.g. a multicore CPU) to a real-time virtual resource. Similar to the period and the execution time for a periodic real-time task, a resource partition also has its own specification. A brief and precise definition of this specification can make it easier to schedule the resource partitions (resource-level scheduling) on the physical resource, as well as to schedule the real-time tasks (task-level scheduling) on the resource partitions. We call the formalization of this specification as the scheduling-interface-definition problem in a hierarchical real-time system. Once a scheduling interface is identified, both the resource-level scheduler and the task-level scheduler can start to work together.

An important factor to consider in the scheduling-interface-definition problem is the type of real-time task models used at the task level. The Regularity-based Resource Partitioning Model [1] is an excellent resource allocation method that works well in tandem with the following widely-adopted task model:

A task T is defined as (c, p), where integer c is the worst case execution time requirement, and integer p is the period and relative deadline of the task. A task can be interrupted or resumed only at integral time instants.

In this task model, the physical resource is allocated in time intervals determined by integral numbers of a time unit, or *quantum* size. Thus, the Regularity-based Resource Partitioning Model considers the scheduling interface in the domain of integers and possesses distinctively interesting properties.

A Regularity-based Resource Partition is specified by its regularity and weight. Mok and Feng [1] present an algorithm (called AAF-Single in [4]) for scheduling such

defined Regularity-based Resource Partitions on a single-resource platform. Then Li and Cheng [4] extend AAF-Single to AAF-Multi for a uniform multiresource platform without violating the previous schedulability bound. In this chapter, we develop new regularity-based resource partitioning algorithms which can significantly achieve better performance than that of AAF-Single and AAF-Multi either on a uniform multiresource platform or on a single-resource one.

3.1 Existing Work

In this section, we review the existing resource-partitioning algorithms introduced [1, 3, 4]. We first introduce irreglular partitions in the RRP Model.

Definition 3.1.1 *Letting a, b, k be non-negative integers, the Supply Regularity $R_s(\Pi)$ of Partition Π is equal to the minimum value of k such that $\forall a, b$, $|Ir(b) - Ir(a)| < k$.*

Definition 3.1.2 *A k Supply-Irregular Partition is a partition with supply regularity of k, where $k > 1$.*

The Regular Partition and the Irregular Partition, especially the former, can provide wonderful real-time support to the lower-level real-time tasks working in the integer domain. We shall not mention them here since we are only concerned with the resource-partitioning problem in this section. The problem we want to solve here is:

Problem I *Given a sequence $\langle(\alpha_i, k_i) : 1 \leq i \leq n\rangle$ as the weights and supply regularities of n resource partitions, schedule them on a physical (single or multiple)*

resource.

Next, we review the algorithms in [3] showing how to solve Problem I on a single-resource platform. The main idea is to convert all resource partitions into a group of regular partitions with weights of powers of one-half whose schedulability can be easily checked on a single-resource platform.

Lemma 3.1.1 *When k regular partitions are combined together, they form a partition with supply regularity of k.*

Proof. Immediately follows from Def. 2.4 and Def. 3.1.1. □

Definition 3.1.3 *Given a Partition* Π *with weight* α *and supply regularity Rs, the Adjusted Availability Factor* $\mathcal{AAF}(\alpha,\ Rs)$ *is the total weight of the partitions that are used to compose* Π.

The process of computing $\mathcal{AAF}$ of a resource partition can be regarded as an approximating function. The basic idea is to find a set of powers of one-half with a cardinality that does not exceed the supply regularity of the partition. Simultaneously, we attempt to keep the sum of these numbers as small as possible. Feng [3] has developed an algorithm for computing $\mathcal{AAF}$; we present a different recursive algorithm that makes more sense:

$$\mathcal{AAF}(\alpha,k)=\begin{cases}0 & if\ \alpha=0;\\ \frac{1}{2^n},\ where\ n=\left\lfloor \log_{\frac{1}{2}}\alpha \right\rfloor & if\ \alpha>0\ and\ k=1;\\ \mathcal{AAF}\left(\alpha-\mathcal{L}(\alpha),k-1\right)+\mathcal{L}(\alpha) & otherwise,\end{cases}$$

where $\mathcal{L}(\alpha) = \frac{1}{2^n}$, *where* $n = \left\lceil \log_{\frac{1}{2}} \alpha \right\rceil$.

For better understandability, we give several examples showing how to compute the $\mathcal{AAF}$ of a partition:

$\mathcal{AAF}(0.17,\ 1) = 0.25,$

$\mathcal{AAF}(0.67,\ 2) = 0.5 + 0.25 = 0.75,$

$\mathcal{AAF}(0.67,\ 3) = 0.5 + 0.125 + 0.0625 = 0.6875,$

$\mathcal{AAF}(0.75,\ 3) = \mathcal{AAF}(0.75,\ 2) = 0.5 + 0.25 = 0.75.$

Theorem 3.1.1 provides the schedulability bound for Problem I given by Mok and Feng [1]. Partitions are schedulable on a single-resource platform if the sum of their $\mathcal{AAF}$ does not exceed 1. The pseudo code of AAF-Single follows. Time units are assigned to the regular partitions from heavy (with a higher weight) to light.

Theorem 3.1.1 [1] *Given a sequence* $\langle(\alpha_i, k_i) : i = 1, 2, ..., n\rangle$ *where* α_i *is the weight and* k_i *is the supply regularity of a partition* P_i*, these partitions are schedulable on a single resource if* $\sum_{i=1}^{n} \mathcal{AAF}(\alpha_i, k_i) \leq 1$.

ALGORITHM AAF-SINGLE

(0) $S := \{P_i : i = 1, 2, ..., n\};$

(1) $A_i := \mathcal{AAF}(\alpha_i, k_i)$ *for all* $i;$

(2) $l := 1;$

(3) **while** $S \neq \phi$ **do**

(4) $w := 1\ /\ 2^l;$

(5) **foreach** $P_j \in S$ **do**

(6) **if** $A_i \geq w$ **then**

(7) *assign time-units to* P_j *at level-*l;

(8) $A_i := A_i - w$;

(9) **if** $A_i = 0$ **then** $S = S - \{P_j\}$ **fi**

(10) **fi**

(11) **od**

(12) l++

(13) **od**

Figure 3.1 shows an example of the outcome from the algorithm AAF-Single for a partition group $\{(0.375,\ 2),\ (0.3125,\ 2),\ (0.3125,\ 2)\}$. Each irregular partition is composed of a number of regular partitions which does not exceed the supply regularity of this irregular partition. Notice that on a regular partition with weight of a power of one-half, the numbers of time units form an arithmetic sequence.

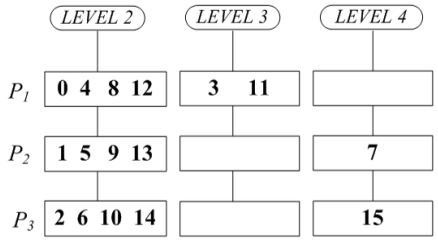


Figure 3.1: AAF-Single

We recently [4] presented a new algorithm called AAF-Multi which extends the schedulability bound given in Theorem 3.1.1 for a single-resource platform to one for a multiresource platform. Theorem 3.1.2 shows this new schedulability bound.

Theorem 3.1.2 [4] *Given a sequence* $\langle(\alpha_i, k_i) : i = 1, 2, ..., n\rangle$ *where* α_i *is the weight and* k_i *is the supply regularity of a partition* P_i*, these partitions are schedulable on* m *resources if* $\sum_{i=1}^{n} \mathcal{AAF}(\alpha_i, k_i) \leq m$.

Suffering from the time-unit overlapping problem, AAF-Single cannot be directly applied onto a multiresource platform. Consider the example in Figure 3.2, where $\{(0.75,\ 2),\ (0.625,\ 2),\ (0.625,\ 2)\}$ are $\mathcal{AAF}$ and supply regularity of P_1, P_2, and P_3. As shown in the top-left part of Figure 3.2, these three partitions cannot be scheduled by AAF-Single because the overlap occurrence on P_3 is unavoidable.

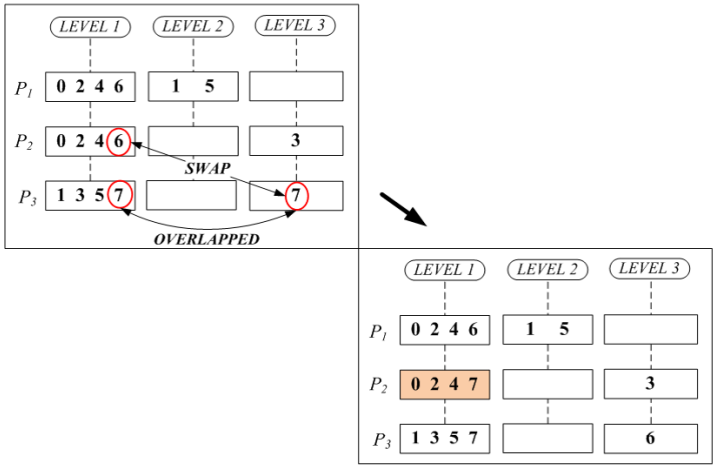


Figure 3.2: From AAF-Single to AAF-Multi

To solve the overlapping problem, we [4] introduced a new type of partition – sub-regular partition. A regular partition with an available factor of a power of

one-half is transformed into a sub-regular partition if a subset of its time units are right-shifted by 1. The first level of P_2 in the bottom-right part of Figure 3.2 is an example of a sub-regular partition, where the time unit 6 is right-shifted to 7. Instead of regular partitions with available factors of powers of one-half, irregular partitions in the partition group are converted into sub-regular partitions in AAF-Multi. We have proven that the supply regularity of a partition composed of k time sub-regular partitions is not greater than k. We then developed a deductive methodology which can schedule these sub-regular partitions together with the regular partitions in the partition group on a multiresource platform.

However, the resource utilization provided by AAF-Single and AAF-Multi is not sufficiently high. In the next two sections, we present new approaches to improve it.

3.2 AAF-Pfair-Mixed: Improvement by Incorporating Pfair

The Pfair scheduling algorithm [5, 6] shares a lot of similarities with the Regularity-based Resource Partitioning Algorithms if we regard resource partitions as periodic real-time tasks. In this section, we shall review some key ideas of Pfair first, and then incorporate Pfair into our resource partitioning algorithms to improve their resource utilization.

3.2.1 Review the Pfair Algorithm

The Pfair algorithm solves the real-time task scheduling problem on a multiprocessor platform by observing the following constraints or rules:

• There are n periodic tasks $\{(e_x, p_x) : 1 \leq x \leq n\}$ to be scheduled over m identical processors, where integer e_x is the worst case execution time, and integer p_x is the period and relative deadline of Task x.

• Tasks are interrupted and resumed only at integral time instants.

• No task can run on more than one processor simultaneously.

• Tasks make progress in a steady rate. Task i receives either $\lfloor w_x \cdot t \rfloor$ or $\lceil w_x \cdot t \rceil$ serving time in the time interval $[0, t]$, where $w_x = e_x/p_x$.

The regularity-based resource partitioning algorithms work in a remarkably similar way as Pfair if we consider regularity-based resource partitions as real-time tasks in Pfair. First, they both work in the domain of integers. All input parameters of tasks and resource partitions are integers. Second, neither allows a task or a resource partition to run concurrently. Last but not least, both consider the difference between the actual resource supply and the ideal resource availability. To explain the last similarity, we cite the following conclusion from [5].

A Schedule S is Pfair if and only if

$$\forall x, t : 1 \leq x \leq n, t \in N : -1 < lag(S, x, t) < 1,$$

where $lag(S, x, t) = w_x \cdot t - \sum_{x=0}^{t-1} S(x, i)$.

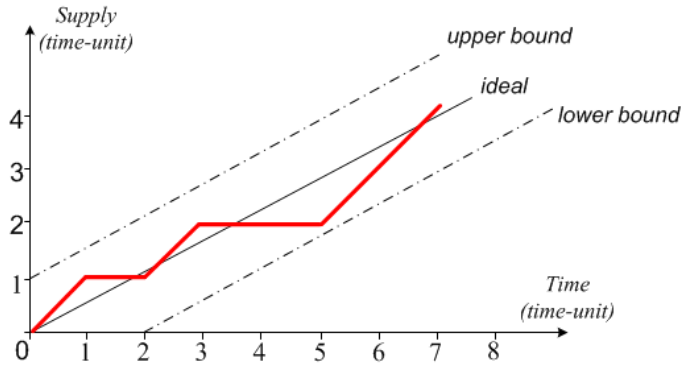


Figure 3.3: A Pfair Task

The value of $\sum_{x=0}^{t-1} S(x,i)$ indicates how many time units Task x has been assigned to by Schedule S at time instant t, which is the same as the Supply Function in Definition 2.2. Furthermore, the definition of function lag shows that its value equals to the minus value of Instant Regularity in Definition 2.3. Because Pfair restricts the value of lag in (-1, 1), if we schedule resource partitions using Pfair, it can be guaranteed that the supply regularity of each partition is 2. This is enough for any irregular partition, but not for any regular partition. Baruah et al. [5] show the schedulability bound of Pfair is 1, which means there is no utilization loss in Pfair. If all of the resource partitions are irregular, we can schedule them with Pfair with 100% resource utilization.

3.2.2 AAF-Pfair-Mixed Algorithm

The investigation on the Pfair algorithm inspires us to improve the current AAF-based resource partitioning algorithms. Our strategy is relatively straightforward –

separating the irregular partitions from the regular ones, as shown in Figure 3.4.

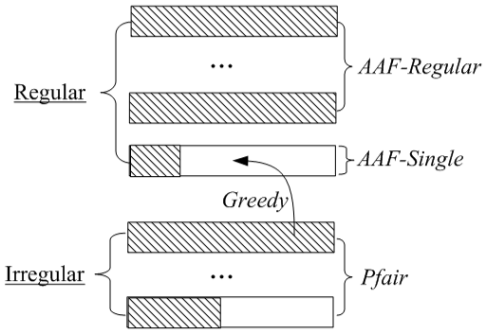


Figure 3.4: AAF-Pfair-Mixed

The resource partitions are grouped into two categories: regular and irregular, which are scheduled on separate resources. A hybrid approach is applied, combining global and partitioned-scheduling strategies. We schedule regular partitions with AAF-Regular and irregular ones with Pfair. This new hybrid algorithm is called AAF-Pfair-Mixed. Next we present the pseudo code of AAF-Regular which schedules a regular-partition set $\{P_i : i = 1, 2, ..., n\}$ on a multiresource, where the weight of P_i is α_i.

ALGORITHM AAF-REGULAR

(0) $S := \{P_i : i = 1, 2, ..., n\}$;

(1) $A_i := \mathcal{AAF}(\alpha_i, 1)$ $for\ all\ i$;

(2) $w := 1; m := 1; r := 1$;

(3) **while** $(S \neq \phi)$ **do**

(4) **foreach** $P_j \in S$ **do**

(5) **if** $(A_i = w)$ **then**

(6) *schedule P_i to the m-th resource*;

(7) $r := r - w$;

(8) **if** $(r = 0)$ **then**

(9) $m{+}{+}$;

(10) $r := 1$;

(11) **fi**

(12) $S = S - \{P_j\}$;

(13) **fi**

(14) **od**;

(15) $w := w/2$;

(16) **od**

By incorporating Pfair, the resource utilization overhead due to the adjusting of the weights of the irregular partitions is removed. However, a new type of overhead emerges as a result of the partitioned-scheduling strategy we have used. As shown in Figure 3.4, partitions in both regular and irregular categories cannot fully utilize the resources assigned to them. In the worst case, nearly 2 resources could be wasted. The overall resource utilization would be drastically impacted especially when the resource number is small. An easy enhancement is migrating some irregular partitions to the last resource belonging to the regular category when it is not fully utilized

(AAF-Regular guarantees that the other resources in the regular category are fully utilized). We can use a simple greedy strategy when choosing irregular partitions for migration. Each time we select the fittest one which can utilize the most capacity of the last resource in the regular category. Notice that we have to adjust the weight of a migrating irregular partition; otherwise, it cannot be scheduled together with the regular partitions sharing the same resource. As a result, AAF-Single can work for the last resource in the regular category.

Another advantage of AAF-Pfair-Mixed is that it can restrict the number of migrating partitions. On one hand, there is no partition migrations in AAF-Regular at all. On the other hand, we can use the enhanced Pfair approach in [7], which guarantees that the number of migrating partitions is less than the number of resources. Although this approach does not necessarily reduce the number of migrations, we can still keep those more important partitions fixed to specific resources, which might be desirable sometimes.

3.3 Magic7: An Optimal Static Approximation Algorithm

Since Pfair cannot schedule regular partitions, we use AAF-Regular to schedule them in the previous section. Are there better solutions than AAF-Regular? We will consider the following problem in this section:

Problem II *Given a sequence* $\langle \alpha_i : 1 \leq i \leq n \rangle$ *as the weights of* n *regular partitions,*

schedule them on a physical (single or multiple) resource.

Problem II is a special case of Problem I when all the resource partitions are regular. Mok and Feng [1] have announced that this is an NP-hard problem, so it is hard to find an optimal scheduling algorithm for all the combinations of weights. However, we can still try to find a generally "good" algorithm.

3.3.1 Static Approximation Algorithm

AAF-Regular is an approximation methodology. It adjusts the weight of each regular partition to the closest greater or equal value in the sequence $\langle 1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, ... \rangle$. We can also choose other sequences like $\langle 1, \frac{1}{3}, \frac{1}{9}, \frac{1}{27}, ... \rangle$ or a finite one $\langle 1, \frac{4}{5}, \frac{3}{5}, \frac{2}{5}, \frac{1}{5} \rangle$. Excluding the fixed element 1, we call such a sequence an Approximating Boundary Sequence (ABS). An ABS satisfies the following:

- *All elements are in the range of* $(0, 1)$.
- *These elements must be strictly decreasing.*
- *If the sequence is infinite, its limit is 0.*

In general, a Static Approximation Algorithm (SAA) adjusts each weight to the closest greater or equal value in a specific ABS. This is tantamount to saying that such an algorithm mostly depends on its ABS. There are some other approximation algorithms depending on more than one ABS, or a dynamic ABS, which might be more efficient but much more complicated. Those algorithms are outside the scope of our work.

Definition 3.3.1 *An Approximating Boundary Sequence (ABS) is a real number sequence* $\langle b_1, b_2, b_3, ... \rangle$, *where* $\forall i > 0,\ 0 < b_{i+1} < b_i < 1$; $\lim b_n = 0$ *if* $n \to \infty$.

Definition 3.3.2 *The Approximating Function of an ABS* $\mathcal{B} = \langle b_1, b_2, b_3, ... \rangle$ *for* $\alpha \in (0, 1)$ *is*

$$\mathcal{R}(\mathcal{B}, \alpha) = b_{i-1} \text{ when } b_i < \alpha \leq b_{i-1},$$

where $b_0 = 1$; $b_{|\mathcal{B}|+1} = 0$ *if* $\mathcal{B}$ *is finite.*

Definition 3.3.3 *An ABS* $\mathcal{B}$ *is feasible if and only if* $\forall \langle \alpha_1, \alpha_2, ..., \alpha_n \rangle$ *as the weights of n time regular partitions, they are schedulable on* $\lceil \sum_{i=1}^{n} \mathcal{R}(\mathcal{B}, \alpha_i) \rceil$ *time resources after being adjusted by the Approximating Function of* $\mathcal{B}$. *For convenience, we derive directly that* $\langle \alpha_1, \alpha_2, ..., \alpha_N \rangle$ *is (un)schedulable under* $\mathcal{B}$.

An SAA is feasible if and only if its ABS is feasible. From now on, we will focus on how to find an optimal ABS. There are three factors that we need to consider. The first is the schedulability bound of such an algorithm. The second is the average resource utilization, which indicates the performance of the algorithm. The third is the feasibility of the ABS we find. We will discuss these three factors one by one in this section.

3.3.2 Schedulability Bound

The schedulability bound of a feasible SAA is the maximum worst-case utilization for which a regular-partition set can be scheduled by this SAA. Mok and Feng [1] have shown that the schedulability bound of the AAF-Regular Algorithm is 0.5.

We will investigate the schedulability bound of general SAAs here. Because the feasibility of an SAA entirely depends on the feasibility of its ABS, we derive the schedulability bound of an ABS directly for convenience. We use $\Upsilon(\mathcal{B})$ to represent the schedulability bound of a feasible ABS $\mathcal{B}$.

Theorem 3.3.1 *If a feasible ABS $\mathcal{B}$ is finite, $\Upsilon(\mathcal{B}) = 0$.*

Proof. Suppose $\mathcal{B} = \langle b_1, b_2, ..., b_n \rangle$, and $\Upsilon(\mathcal{B}) = \epsilon \in (0, 1]$. Let $N = \lfloor \frac{1}{b} \rfloor + 1 > \frac{1}{b}$, where $b = b_n$ or $b = 1$ if $\mathcal{B}$ is empty. $\langle \alpha_i = \frac{b \cdot \epsilon}{N} : i = 1, 2, ..., N \rangle$ is unschedulable under $\mathcal{B}$ on a single-resource because $\sum_{i=1}^{N} \mathcal{R}(\mathcal{B}, \alpha_i) = N \cdot b > 1$. However, its overall utilization equals to $N \cdot \frac{b \cdot \epsilon}{N} \leq \epsilon$, which contradicts $\Upsilon(\mathcal{B}) = \epsilon$. □

Theorem 3.3.2 *If a feasible ABS $\mathcal{B} = \langle b_1, b_2, b_3, ... \rangle$ is infinite, $\Upsilon(\mathcal{B}) = \min\left\{ \frac{b_{i+1}}{b_i} : i \geq 0 \right\}$, where $b_0 = 1$.*

Proof. Let $\epsilon = \min\left\{ \frac{b_{i+1}}{b_i} : i \geq 0 \right\}$.

• Suppose $\langle \alpha_i : i \in [1, n] \rangle$ is the worst-case weight combination to be scheduled on m time resources. Then $\forall \rho \in (0, 1), \sum_{i=1}^{n} \mathcal{R}(\mathcal{B}, \alpha_i) > \rho \cdot m$ since $\mathcal{B}$ is infinite.

Suppose $\alpha_i \in (b_{k_i+1}, b_{k_i}]$, then

$$\alpha_i > b_{k_i+1} \geq b_{k_i} \cdot \epsilon = \mathcal{R}(\mathcal{B}, \alpha_i) \cdot \epsilon.$$

So $\forall \rho \in (0, 1), \sum_{i=1}^{n} \alpha_i = \sum_{i=1}^{n} \mathcal{R}(\mathcal{B}, \alpha_i) > \rho \cdot \epsilon \cdot m$, which concludes $\Upsilon(\mathcal{B}) \geq \epsilon$.

• If $\Upsilon(\mathcal{B}) > \epsilon$, suppose $\Upsilon(\mathcal{B}) = (1 + 2\delta) \cdot \epsilon$ where $\delta \in \left(0, \frac{1-\epsilon}{2\epsilon}\right]$.

$\exists k, \frac{b_{k+1}}{b_k} < \sqrt{1 + \delta} \cdot \epsilon$.

Let $\alpha = \sqrt{1+\delta} \cdot b_{k+1} < (1+\delta) \cdot \epsilon \cdot b_k \leq (1 + \frac{1-\epsilon}{2\epsilon}) \cdot \epsilon \cdot b_k < b_k$.

Let $m = \left\lfloor \frac{b_k \cdot (1+\delta)}{\delta} \right\rfloor + 1 > \frac{b_k \cdot (1+\delta)}{\delta}$.

Let $n = \left\lfloor \frac{m}{b_k} \right\rfloor + 1 \in \left(\frac{m}{b_k}, \frac{m}{b_k} + 1 \right]$.

The sequence $\langle \alpha_i = \alpha : i = 1, 2, ..., n \rangle$ is unschedulable by $\mathcal{B}$ on m time resources because $n \cdot \mathcal{R}(\mathcal{B}, \alpha) = n \cdot b_k > \frac{m}{b_k} \cdot b_k = m$.

However, its overall utilization is

$$
\begin{aligned}
& n \cdot \sqrt{1+\delta} \cdot b_{k+1} \\
< & (\tfrac{m}{b_k} + 1) \cdot (1+\delta) \cdot \epsilon \cdot b_k \\
= & (m + b_k) \cdot (1+\delta) \cdot \epsilon \\
= & m \cdot (1+\delta) \cdot \epsilon + b_k \cdot (1+\delta) \cdot \epsilon \\
< & m \cdot (1+\delta) \cdot \epsilon + m \cdot \delta \cdot \epsilon \\
= & m \cdot (1+2\delta) \cdot \epsilon,
\end{aligned}
$$

which contradicts $\Upsilon(\mathcal{B}) = (1+2\delta) \cdot \epsilon$. □

3.3.3 Average Resource Utilization

Besides the schedulability bound, the average resource utilization is another important aspect of a scheduling algorithm. The approximating process leads to some utilization overhead, the ratio of which determines the performance of an SAA. As

shown in Figure 3.5, the polygonal line with a shape of stairs indicates the Approximating Function of an ABS $\mathcal{B} = \langle b_1, b_2, b_3, ... \rangle$, and the overall area of the slash filled triangles indicates the utilization overhead of the approximating process by $\mathcal{B}$.

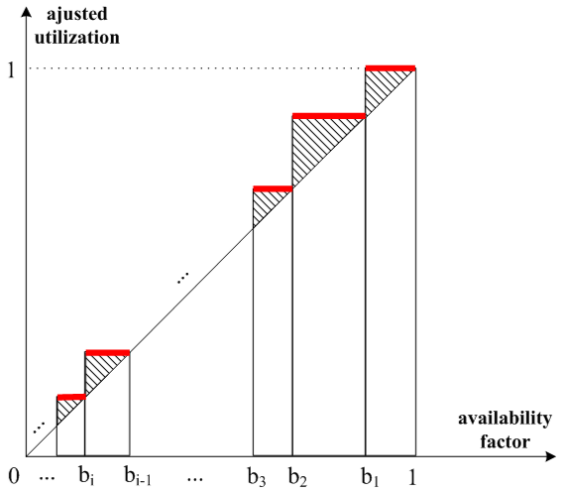


Figure 3.5: The Approximating Function and its Overhead

Definition 3.3.4 *The Utilization Overhead of an ABS* $\mathcal{B} = \langle b_1, b_2, b_3, ... \rangle$ *is*

$$\mathcal{O}(\mathcal{B}) = \frac{1}{2} \sum_{i \geq 0} (b_i - b_{i+1})^2 \in \left(0, \frac{1}{2}\right],$$

where $b_0 = 1$*;* $b_{|\mathcal{B}|+1} = 0$ *if* $\mathcal{B}$ *is finite.*

Finally, the average resource utilization of $\mathcal{B}$ equals to $\frac{0.5}{\mathcal{O}(\mathcal{B})+0.5} \in \left[\frac{1}{2}, 1\right)$, which is decreasing when $\mathcal{O}(\mathcal{B})$ is increasing. The average resource utilization equals the minimum value 0.5 when $\mathcal{B}$ is empty, where a dedicated resource is assigned to each regular partition.

3.3.4 Feasible Approximating Boundary Sequences

First, we can deride the Schedulability Bound of feasible ABSes as follows.

Theorem 3.3.3 $\Upsilon(\mathcal{B}) \leq \frac{1}{2}$, *where* $\mathcal{B}$ *is any feasible ABS.*

Proof. Suppose $\mathcal{B} = \langle b_1, b_2, b_3, ... \rangle$, where $b_1 = \frac{q}{p}$ and p, q are co-prime. Then $\exists m, n > 0, m \cdot p = n \cdot q + 1$.

If $\Upsilon(\mathcal{B}) > \frac{1}{2}$, by Theorem 3.3.2, $\forall i \geq 0, \frac{b_{i+1}}{b_i} > \frac{1}{2}$, where $b_0 = 1$, then $p \geq 3$ and $\exists j, b_j \in \left(\frac{1}{2p}, \frac{1}{p}\right)$.

Let's consider the problem whether the sequence $\langle b_1, b_1, ..., b_1, b_j \rangle$ with n times b_1 elements is schedulable under $\mathcal{B}$. Since $n \cdot b_1 = m - \frac{1}{p}$, if it is schedulable on m resources, the regular partition with availability factor b_j must be a part of a regular partition with availability factor $\frac{1}{p}$. This is impossible by Lemma 2.4. □

Same as Problem II, the decision problem of the feasibility of an arbitrary ABS is very hard. Hereby, we start from investigating several special types of ABSes. The first is the Geometric Approximating Boundary Sequence. The ABS of the AAF-Regular algorithm can belong in this type.

Definition 3.3.5 *A Geometric Approximating Boundary Sequence* $\mathcal{G}(m)$ *is specified as the infinite geometric sequence* $\left\langle \frac{1}{m}, \frac{1}{m^2}, \frac{1}{m^3}, ... \right\rangle$ *where integer* $m > 1$.

Theorem 3.3.4 $\forall m > 1, \mathcal{G}(m)$ *is feasible.*

Proof. We use a similar approach as with AAF-Regular. □

Another interesting type of ABS is the Arithmetic Approximating Boundary Sequence. This is a kind of finite ABS.

Definition 3.3.6 *An Arithmetic Approximating Boundary Sequence* $\mathcal{A}(n)$ *is specified as the finite arithmetic sequence* $\left\langle \frac{n-1}{n}, \frac{n-2}{n}, ..., \frac{1}{n} \right\rangle$ *where integer* $n > 1$.

To find feasible Arithmetic Approximating Boundary Sequences, we introduce the Regularity Magic Number (RMN). Regular partitions can be merged and split if their periods are all equal to a common RMN.

Definition 3.3.7 *Integer* $\mathcal{P}$ *is a Regularity Magic Number (RMN) if and only if* $\forall q_1, q_2 \in (0, \mathcal{P})$ *where* $q_1 + q_2 \leq \mathcal{P}$, $\exists \delta_1, \delta_2 \in [0, \mathcal{P})$, *where*

$$T_0(\mathcal{P}, q_1 + q_2) = T(\mathcal{P}, q_1, \delta_1) \cup T(\mathcal{P}, q_2, \delta_2).$$

Corollary 3.3.1 *Given a Regularity Magic Number* $\mathcal{P}$, $\forall q_1, q_2$, *where* $0 < q_2 < q_1 \leq \mathcal{P}$, $\exists \delta_1, \delta_2 \in [0, \mathcal{P})$, *where*

$$T_0(\mathcal{P}, q_1 - q_2) = T(\mathcal{P}, q_1, \delta_1) - T(\mathcal{P}, q_2, \delta_2).$$

Theorem 3.3.5 *The set of all Regularity Magic Numbers is* $\{2, 3, 4, 5, 7\}$.

Proof. We can check that 2, 3, 4, 5, 7 are RMNs. For example, 4 is an RMN since:

$T_0(4, 1) = \langle 0 \rangle$;

$T_0(4, 2) = \langle 0, 2 \rangle = T_0(4, 1) \cup T(4, 1, 2)$;

$T_0(4, 3) = \langle 0, 1, 2 \rangle = T(4, 1, 1) \cup T_0(4, 2)$;

$T_0(4, 4) = \langle 0, 1, 2, 3 \rangle = T(4, 1, 3) \cup T_0(4, 3)$;

$T_0(4,4) = T_0(4,2) \cup T(4,2,1)$.

There is no other RMN since $\forall n \geq 6$ and $n \neq 7, T(n,2,\delta) \subset T_0(n,3)$ is impossible by Lemma 2.4. □

Theorem 3.3.6 *$\mathcal{A}(n)$ is feasible if and only if n is a Regularity Magic Number.*

Proof. First $\mathcal{A}(n)$ is infeasible when n is not an RMN because $\langle \frac{n-3}{n}, \frac{2}{n} \rangle$ is unschedulable under $\mathcal{A}(n)$ on a single-resource ($T(n,2,\delta) \subset T_0(n,3)$ is impossible by Lemma 2.4).

Then we use an inductive approach to prove that $\mathcal{A}(n)$ is feasible when n is an RMN. First, $\mathcal{A}(n)$ is feasible for a single-resource by the definition of RMN. Suppose $\mathcal{A}(n)$ is feasible when the number of the resources is less than M.

Given $\langle \frac{q_1}{n}, \frac{q_2}{n}, ..., \frac{q_m}{n} \rangle$, where $\sum_{i=1}^{m} \frac{q_i}{n} = M$, as the weights of regular partitions $P_1, P_2, ..., P_m$. We will show that they are schedulable on M resources.

Find k, where $\sum_{i=1}^{k} \frac{q_i}{n} \leq 1$ and $\sum_{i=1}^{k+1} \frac{q_i}{n} > 1$.

If $\sum_{i=1}^{k} \frac{q_i}{n} = 1$, these partitions can be easily scheduled: $P_1, P_2, ..., P_k$ on one resource and $P_{k+1}, ..., P_m$ on the other $M-1$ resources.

Otherwise $\sum_{i=1}^{k} \frac{q_i}{n} < 1$ and $\sum_{i=1}^{k+1} \frac{q_i}{n} > 1$. Let $\frac{q}{n} = \sum_{i=1}^{k+1} \frac{q_i}{n} - 1 \in \left(0, \frac{q_{k+1}}{n}\right)$, then $\langle \frac{q}{n}, \frac{q_{k+2}}{n}, ..., \frac{q_m}{n} \rangle$ can be scheduled on $M-1$ resources. Suppose the time-unit sequence of the first partition is $T(n,q,\delta)$. By Corollary 3.3.1, $\exists \delta_1, \delta_2$, where

$$T_0(n,q) = T(n, q_{k+1}, \delta_1) - T(n, q_{k+1} - q, \delta_2),$$

then

$$T(n, q, \delta) = T(n, q_{k+1}, \delta_1 + \delta) - T(n, q_{k+1} - q, \delta_2 + \delta).$$

At last, we just need to schedule $\left\langle \frac{q_1}{n}, \frac{q_2}{n}, ..., \frac{q_{k+1}-q}{n} \right\rangle$ on the left resource and let the last partition have the time-unit sequence $T(n, q_{k+1} - q, \delta_2 + \delta)$. □

Since the schedulability bound of an Arithmetic ABS is zero by Theorem 3.3.1, we define the third type of ABS – the Hybrid Approximating Boundary Sequence, which is infinite but can inherit the feasibility property from the Arithmetic ABS.

Definition 3.3.8 *A Hybrid Approximating Boundary Sequence* $\mathcal{H}(n, m)$ *is specified as the sequence*

$\left\langle \frac{n-1}{n}, \frac{n-2}{n}, ..., \frac{1}{n}, \frac{1}{n \cdot m}, \frac{1}{n \cdot m^2}, \frac{1}{n \cdot m^3}, ... \right\rangle$ *where integers* $n > 1$, $m > 1$.

Theorem 3.3.7 $\mathcal{H}(n, m)$ *is feasible if and only if* n *is a Regularity Magic Number.*

Proof. : The scheduling has two steps. First schedule the elements in $\left\{1, \frac{n-1}{n}, \frac{n-2}{n}, ..., \frac{1}{n}\right\}$ using the strategy of scheduling $\mathcal{A}(n)$. Then schedule the other elements on the remaining part of the resources using the strategy of scheduling $\mathcal{G}(m)$. □

$\mathcal{B}$	$\Upsilon(\mathcal{B})$	$\mathcal{O}(\mathcal{B})$	Feasibility
$\mathcal{G}(m)$	$\frac{1}{m}$	$\frac{1}{2} - \frac{1}{m+1}$	feasible
$\mathcal{A}(n)$	0	$\frac{1}{2n}$	feasible iif n is an RMN
$\mathcal{H}(n, m)$	$\frac{1}{m}$	$\frac{1}{2n} - \frac{1}{n^2(m+1)}$	feasible iif n is an RMN

Table 3.1: A Comparison Between Three Types of ABS

Table 3.1 shows a comparison of the three types of ABS we have defined in this subsection. We can conclude from it that the smallest value of m and the largest

value of n can lead to the best performance. Theorem 3.3.7 shows the maximum Regularity Magic Number is 7, so we find that the optimal ABS so far is $\mathcal{H}(7,2)$.

3.3.5 Extended Approximating Boundary Sequence

To achieve the optimal resource utilization, we extend the concept of Approximating Boundary Sequence.

Definition 3.3.9 *An Extended Approximating Boundary Sequence (E-ABS) is a tuple* $(\mathcal{B},\mathcal{B}')$*, where* $\mathcal{B}=\langle b_1,b_2,b_3,...\rangle$*,* $\mathcal{B}'=\langle b'_1,b'_2,b'_3,...\rangle$ *are two ABSes;* $b_1+b'_1<1$ *if neither* $\mathcal{B}$ *nor* $\mathcal{B}'$ *is empty.*

The boundary sequence of the upper E-ABS can be regarded as $\langle ...,1-b'_3,1-b'_2,1-b'_1,b_1,b_2,b_3,...\rangle$. An ABS $\mathcal{B}$ corresponds to an E-ABS $(\mathcal{B},\langle\rangle)$.

Definition 3.3.10 *The Approximating Function of an E-ABS* $\mathcal{E}=(\mathcal{B}=\langle b_1,b_2,b_3,...\rangle,$ $\mathcal{B}'=\langle b'_1,b'_2,b'_3,...\rangle)$ *for* $\alpha\in(0,1)$ *is*

$$\mathcal{R}(\mathcal{E},\alpha)=\begin{cases}\mathcal{R}(\mathcal{B},\alpha) & if\ \alpha\le b;\\ 1-b'_{i+1}\ when\ \alpha\in(1-b'_i,1-b'_{i+1}] & otherwise,\end{cases}$$

where $b=b_1:0\ ?\ |\mathcal{B}|>0$*;* $b'_0=1-b$*;* $b'_{|\mathcal{B}'|+1}=0$ *if* $\mathcal{B}'$ *is finite.*

Same as a feasible ABS, $\Upsilon(\mathcal{E})$ is specified as the Schedulability Bound of a feasible E-ABS $\mathcal{E}$.

Theorem 3.3.8 *Given a feasible E-ABS* $\mathcal{E}=(\mathcal{B}=\langle b_1,b_2,b_3,...\rangle,\mathcal{B}'=\langle b'_1,b'_2,b'_3,...\rangle)$,

$$\Upsilon(\mathcal{E})=\min\left\{\Upsilon(\mathcal{B}),\min\left\{\tfrac{1-b'_{i-1}}{1-b'_i}:i>0\right\}\right\},$$

where $b'_0 = 1 - b_1 : 1\ ?\ |\mathcal{B}| > 0$*;* $b'_{|\mathcal{B}'|+1}$=0 *if* $\mathcal{B}'$ *is finite.*

Definition 3.3.11 *The Utilization Overhead of an E-ABS* $\mathcal{E} = (\mathcal{B} = \langle b_1, b_2, b_3, ...\rangle, \mathcal{B}' = \langle b'_1, b'_2, b'_3, ...\rangle)$ *is*

$\mathcal{O}(\mathcal{E}) = \frac{1}{2}(\sum_{i>0}(b_i - b_{i+1})^2 + (1 - b_1 - b'_1)^2 + \sum_{i>0}(b'_i - b'_{i+1})^2)$,

where $b_{|\mathcal{B}|+1} = 0$ *if* $\mathcal{B}$ *is finite;* $b_{|\mathcal{B}'|+1} = 0$ *if* $\mathcal{B}'$ *is finite.*

Corollary 3.3.2 *Given an E-ABS* $\mathcal{E} = (\mathcal{B}, \mathcal{B}')$ *where neither* $\mathcal{B}$ *nor* $\mathcal{B}'$ *is non-empty,*

$$\mathcal{O}(\mathcal{E}) < \min\{\mathcal{O}(\mathcal{B}), \mathcal{O}(\mathcal{B}')\}.$$

Definition 3.3.12 *Given* $\alpha \in (0, 1)$ *and an E-ABS* $\mathcal{E} = (\mathcal{B}, \mathcal{B}')$*,* $\alpha \in \mathcal{E}$ *if and only if* $\alpha \in \mathcal{B}$ *or* $1 - \alpha \in \mathcal{B}'$*.*

Theorem 3.3.9 $(\mathcal{H}(n, m), \langle \frac{1}{n \cdot m}, \frac{1}{n \cdot m^2}, \frac{1}{n \cdot m^3}, ...\rangle)$ *is feasible if and only if* n *is a Regularity Magic Number.* $\mathcal{Z}(n, m)$ *is specified as such an E-ABS.*

Proof. $\mathcal{Z}(n, m)$ is infeasible if n is not an RMN (by Theorem 3.3.6).

Then we use an inductive approach to prove that $\mathcal{Z}(n, m)$ is feasible if n is an RMN. First, $\mathcal{Z}(n, m)$ is feasible for a single-resource obviously. Suppose $\mathcal{Z}(n, m)$ is feasible when the number of the resources is less than M.

Given a non-increasing sequence $Q = \langle \alpha_1, \alpha_2, ..., \alpha_K \rangle$, where $\sum_{i=1}^{K} \alpha_i = M$ and $\alpha_i \in \mathcal{Z}(n, m) : i = 1, 2, ..., K$, as the weights of regular partitions $P_1, P_2, ..., P_K$, we will show that they are schedulable on M resources.

- if $\alpha_1 \leq \frac{n-1}{n}$, Q is schedulable by Theorem 3.3.7.
- if $\alpha_1 > \frac{n-1}{n}$ and $\alpha_i = \alpha_1 : i = 2, 3, ..., m$, suppose $\alpha_1 = 1 - \frac{1}{n \cdot m^e}$ where $e > 0$.

Let $\alpha = m \cdot \alpha_1 - (m-1) = 1 - \frac{1}{n \cdot m^{e-1}}$. Then $\langle \alpha, \alpha_{m+1}, \alpha_{m+2}, ..., \alpha_K \rangle$ is schedulable on $M-m+1$ resources. Suppose the time-unit sequence of the first partition is

$$T_0(n \cdot m^{e-1}, n \cdot m^{e-1}) - T(n \cdot m^{e-1}, 1, \delta).$$

At last, we just need to schedule $P_1, P_2, ..., P_m$ like this: for $i = 1, 2, ..., m$, the time-unit sequence of P_i is

$$T_0(n \cdot m^e, n \cdot m^e) - T(n \cdot m^e, 1, \delta + (i-1) \cdot n \cdot m^{e-1}).$$

• Otherwise, because $\sum_{i=1}^{K} \alpha_i = M$, $\exists k_1, k_2, ..., k_J, \sum_{i=1}^{J} \alpha_{k_i} = 1 - \alpha_1$.

Then we can schedule $P_1, P_{k_1}, P_{k_2}, ..., P_{k_J}$ on one resource, and the other partitions on the other $M-1$ resources. □

Lemma 3.3.1 *If an RBS* $\mathcal{B} = \langle b_1, b_2, b_3, ... \rangle$ *is feasible, where* $\Upsilon(\mathcal{B}) = \frac{1}{2}; b_k \geq \frac{1}{2}; \forall n > 0, b_k \neq 1 - \frac{2n+1}{4n+3}$, *then*

$$1 - b_k \in \mathcal{B};\ \tfrac{1-b_k}{2} \in \mathcal{B};\ \forall i, b_i \notin \left(\tfrac{1-b_k}{2}, 1 - b_k\right).$$

Proof. If $\exists i, b_i \in \left(\frac{1-b_k}{2}, 1 - b_k\right)$, by Lemma 2.4, $\langle b_k, b_i \rangle$ is unschedulable under $\mathcal{B}$ on a single resource. Then $1 - b_k \in \mathcal{B}$ and $\frac{1-b_k}{2} \in \mathcal{B}$ because $\Upsilon(\mathcal{B}) = \frac{1}{2}$ (by Theorem 3.3.2). □

Lemma 3.3.2 *If an RBS* $\langle b_1, b_2, b_3, ... \rangle$ *is feasible, where* $\Upsilon(\mathcal{B}) = \frac{1}{2}; b_i > b_j \geq \frac{1}{2}; 1 - b_j < 2(1 - b_i)$, *then*

$$\exists n > 0,\ b_i = 1 - \tfrac{n+1}{4n+3}; b_j = 1 - \tfrac{2n+1}{4n+3}.$$

Proof. • Case 1: $1 - b_i \in \mathcal{B}$

Because $1 - b_i < 1 - b_j < 2(1 - b_i)$ and $(1 - b_i) + (1 - b_j) < 1$, by Lemma 2.4, $1 - b_i = \frac{n+1}{4n+3}$ and $1 - b_j = \frac{2n+1}{4n+3}$ (otherwise, $\langle b_j, 1 - b_i \rangle$ is unschedulable under $\mathcal{B}$ on a single resource).

• Case 2: $1 - b_i \notin \mathcal{B}$

By Lemmas 3.3.4 and 3.3.5, $\exists m > 0, b_i = 1 - \frac{2m+1}{4m+3}$ and $\frac{m+1}{4m+3} \in \mathcal{B}$.

Because $\frac{m+1}{4m+3} < 1 - b_j \leq \frac{1}{2} < 2 \cdot \frac{m+1}{4m+3}$ and $1 - b_j \neq 1 - b_i = \frac{2m+1}{4m+3}$, by Lemma 2.4, $\langle b_j, \frac{m+1}{4m+3} \rangle$ is unschedulable under $\mathcal{B}$ on a single resource. □

Lemma 3.3.3 *If an RBS $\langle b_1, b_2, b_3, ... \rangle$ is feasible, where $\Upsilon(\mathcal{B}) = \frac{1}{2}$, it is impossible that $\exists i, j, \frac{1}{3} \leq b_i < b_j \leq \frac{1}{2}$.*

Proof. Otherwise, $\langle b_i, b_j \rangle$ is unschedulable under $\mathcal{B}$ on a single resource by Lemma 2.4. □

Definition 3.3.13 *Given two E-ABSes $\mathcal{E}_1$ and $\mathcal{E}_2$, $\mathcal{E}_1 = \mathcal{E}_2$ if and only if $\forall \alpha \in \mathcal{E}_1, \alpha \in \mathcal{E}_2$ and $\forall \alpha \in \mathcal{E}_2, \alpha \in \mathcal{E}_1$.*

Theorem 3.3.10 *If a feasible E-RBS $\mathcal{E} \neq \mathcal{Z}(7, 2)$ and $\Upsilon(\mathcal{E}) = \frac{1}{2}, \mathcal{O}(\mathcal{E}) > \mathcal{O}(\mathcal{Z}(7, 2))$.*

Proof. $\mathcal{O}(\mathcal{Z}(7, 2)) = \frac{17}{294} < 0.058$.

Suppose $\mathcal{E} = (\mathcal{B} = \langle b_1, b_2, b_3, ... \rangle, \mathcal{B}' = \langle b'_1, b'_2, b'_3, ... \rangle)$.

Since we can transfer elements between $\mathcal{B}$ and $\mathcal{B}'$, without loss of generality, we assume $\forall b_i \in \mathcal{B}, b_i < \frac{1}{2}$; $\forall b'_i \in \mathcal{B}', b'_i \leq \frac{1}{2}$.

Case 1: $|\mathcal{B}'| \leq 1$

$\mathcal{O}(\mathcal{E}) \geq (\frac{1}{4})^2 > 0.06 > \mathcal{O}(\mathcal{Z}(7,2))$.

Case 2: $|\mathcal{B}'| > 1$

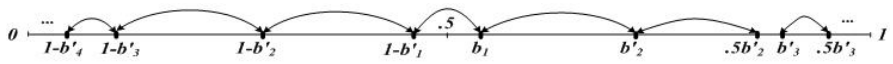
• Case 2-1: $\exists i < j,\ b'_i < 2b'_j$

By Lemma 3.3.2, $b'_i = \frac{2n+1}{4n+3}; b'_j = \frac{n+1}{4n+3}$.

Still by Lemma 3.3.2, $i = 1; j = 2; \forall k \geq 2, b'_k \geq 2b'_{k+1}$.

By Lemma 3.3.1, $\forall k \geq 2, b'_k \in \mathcal{B};\ \frac{b'_k}{2} \in \mathcal{B};\ \forall l, b_l \notin \left(\frac{b'_k}{2}, b'_k\right)$.

By Lemma 2.4, if $b_l > \frac{n+1}{4n+3}, b_l = \frac{2n+1}{4n+3}$.



Thus, $2\mathcal{O}(\mathcal{E}) \geq \sum_{k\geq 2}(b'_k - b'_{k+1})^2 + (\frac{n}{4n+3})^2 + (\frac{1}{4n+3})^2 + (\frac{n}{4n+3})^2 + \sum_{k\geq 2}(b'_k - \frac{b'_k}{2})^2$.

$F(x)$ denotes the lower bound of $\sum_{k\geq 0}(x_k - x_{k+1})^2 + \frac{1}{4}\sum_{k\geq 0}(x_k)^2$, where $x_0 = x \in \left(0, \frac{1}{2}\right)$ and $\forall k \geq 0, x_{k+1} \in \left(0, \frac{x_k}{2}\right]$.

Then $\forall k \geq 0, F(x_k) = F(x_{k+1}) + (x_k - x_{k+1})^2 + \frac{(x_k)^2}{4}$ and $F(x_{k+1}) = (\frac{x_{k+1}}{x_k})^2 F(x_k)$

$\Rightarrow \frac{4}{x_k} + \frac{F(x_k)}{(x_k)^2} = \frac{9x_k - 8x_{k+1}}{(x_k + x_{k+1})(x_k - x_{k+1})}$, where $x_k \in \left(0, \frac{1}{2}\right)$ and $x_{k+1} \in \left(0, \frac{x_k}{2}\right]$

$\Rightarrow \forall k \geq 0, x_{k+1} = \frac{x_k}{2}$

$\Rightarrow F(x) = \frac{2}{3}x^2$.

Thus, $2\mathcal{O}(\mathcal{E}) \geq F(b'_2) + \frac{2n^2+1}{(4n+3)^2} = \frac{2}{3} \cdot (\frac{n+1}{4n+3})^2 + \frac{2n^2+1}{(4n+3)^2} = \frac{8n^2+4n+5}{3(4n+3)^2}$, where $n \geq 1$

$\Rightarrow \mathcal{O}(\mathcal{E}) \geq \frac{17}{294}$.

$\mathcal{O}(\mathcal{E})$ equals to $\frac{17}{294}$ only when $n = 1, b'_1 = \frac{3}{7}, b'_2 = \frac{2}{7}$ and

$\mathcal{E} = \left(\left\langle \frac{3}{7}, \frac{2}{7}, \frac{1}{7}, \frac{1}{14}, \frac{1}{28}, ... \right\rangle, \left\langle \frac{3}{7}, \frac{2}{7}, \frac{1}{7}, \frac{1}{14}, \frac{1}{28}, ... \right\rangle\right) = \mathcal{Z}(7,2)$.

• Case 2-2: $\forall i < j,\ b'_i \geq 2b'_j$

By Lemma 3.3.1, $\forall k \geq 1, b'_k \in \mathcal{B};\ \frac{b'_k}{2} \in \mathcal{B};\ \forall l, b_l \notin \left(\frac{b'_k}{2}, b'_k\right)$.

Case 2-2-1: $b'_1 \leq \frac{1}{3}$

By Lemma 3.3.3, There is at most one element of $\mathcal{B}$ in $\left(\frac{1}{3}, \frac{1}{2}\right)$.

Then: $2\mathcal{O}(\mathcal{E}) > F(b'_1) + (1 - b'_1 - \frac{1}{2})^2 + (\frac{1}{2} - \frac{1}{3})^2 = \frac{2}{3}(b'_1)^2 + (b'_1)^2 - b'_1 + \frac{5}{18} \geq \frac{23}{180} > 0.127 > 2\mathcal{O}(\mathcal{Z}(7,2))$.

Case 2-2-2: $b'_1 > \frac{1}{3}$

By Lemma 3.3.3, b'_1 is the only element of $\mathcal{B}$ in $\left(\frac{1}{3}, \frac{1}{2}\right]$.

Then: $2\mathcal{O}(\mathcal{E}) \geq F(b'_1) + (1 - b'_1 - b'_1)^2 = \frac{2}{3}(b'_1)^2 + (1 - 2b'_1)^2 \geq \frac{1}{7} > 0.142 > 2\mathcal{O}(\mathcal{Z}(7,2))$. □

Finally, we derive that $\mathcal{Z}(7,2)$ is the optimal E-ABS among those which keep the maximum schedulability bound. Magic7 is named for the Static Approximation Algorithm with $\mathcal{Z}(7,2)$, whose approximation boundary sequence is regarded as $\left\langle ..., \frac{55}{56}, \frac{27}{28}, \frac{13}{14}, \frac{6}{7}, \frac{5}{7}, \frac{4}{7}, \frac{3}{7}, \frac{2}{7}, \frac{1}{7}, \frac{1}{14}, \frac{1}{28}, \frac{1}{56}, ... \right\rangle$. Table 3.2 compares Magic7 and AAF-Regular.

	$\mathcal{B}$	$\Upsilon(\mathcal{B})$	$\mathcal{O}(\mathcal{B})$	Average Utilization
AAF-Regular	$\mathcal{G}(2)$	0.5	0.167	75%
Magic7	$\mathcal{Z}(7,2)$	0.5	0.058	89.6%

Table 3.2: A Comparison between Magic7 and AAF-Regular

3.3.6 Magic7-Single and Magic7-Pfair-Mixed

Magic7 can be incorporated into the solutions for Problem I which schedules both regular and irregular partitions. Magic7-Single is an enhanced version of AAF-Single, whose Approximating Function is $\mathcal{AAF}'(\alpha, k)$.

$$\mathcal{AAF}'(\alpha,k) = \begin{cases} 0 & if\ \alpha = 0; \\ \frac{1}{7\cdot 2^n},\ where\ n = \left\lfloor \log_{\frac{1}{2}} 7\alpha \right\rfloor & if\ k{=}1, \alpha{\in}\left(0, \frac{1}{7}\right); \\ \frac{\lceil 7\alpha \rceil}{7} & if\ k{=}1, \alpha{\in}\left[\frac{1}{7}, \frac{6}{7}\right]; \\ 1{-}\frac{1}{7\cdot 2^n},\ where\ n = \left\lceil \log_{\frac{1}{2}} 7(1-\alpha) \right\rceil & if\ k{=}1, \alpha{\in}\left(\frac{6}{7}, 1\right); \\ \mathcal{AAF}'(\alpha - \mathcal{L}'(\alpha), k-1) + \mathcal{L}'(\alpha) & otherwise, \end{cases}$$

$$where\ \mathcal{L}'(\alpha) = \begin{cases} \frac{1}{7\cdot 2^n},\ where\ n{=}\left\lceil \log_{\frac{1}{2}} 7\alpha \right\rceil & if\ \alpha{\in}\left(0, \frac{1}{7}\right); \\ \frac{\lfloor 7\alpha \rfloor}{7} & if\ \alpha{\in}\left[\frac{1}{7}, \frac{6}{7}\right]; \\ 1{-}\frac{1}{7\cdot 2^n},\ where\ n{=}\left\lfloor \log_{\frac{1}{2}} 7(1-\alpha) \right\rfloor & if\ \alpha{\in}\left(\frac{6}{7}, 1\right); \\ 1 & if\ \alpha = 1. \end{cases}$$

Correspondingly, Magic7-Pfair-Mixed is an enhanced version of AAF-Pfair-Mixed, as shown in Figure 3.6. Besides improving the resource utilization, Magic7-Pfair-Mixed also guarantees that the number of the migrating partitions is less than that of the resources, and each migrating partition preempts time units on at most two resources.

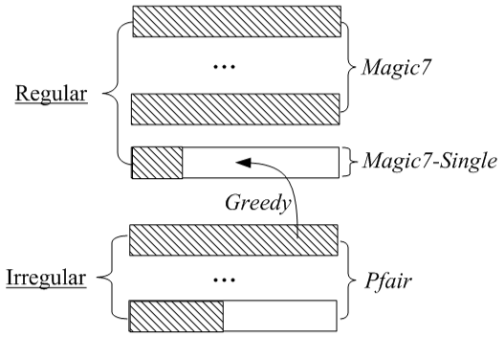


Figure 3.6: Magic7-Pfair-Mixed

3.4 Simulation

We generate regular or irregular partitions whose weights are randomly chosen in $(0, 1)$, and supply regularities are randomly chosen in $[1, MaxReg]$, where $MaxReg$ is a varying parameter. In Figures 3.7–3.9, the horizontal axes represent the utilization of the randomly generated partition sets, and the vertical axes represent the schedulability of these partition sets by different algorithms. From the charts, we can see that the Magic7-enhanced algorithm improves the resource utilization remarkably in each scenario.

3.5 Contribution and Future Work

We investigate the resource partitioning problems (Problem I and Problem II) in the Regularity-based Resource Partition Model. Our contributions include:

• We present a hybrid approach which separates the two types of regularity-based

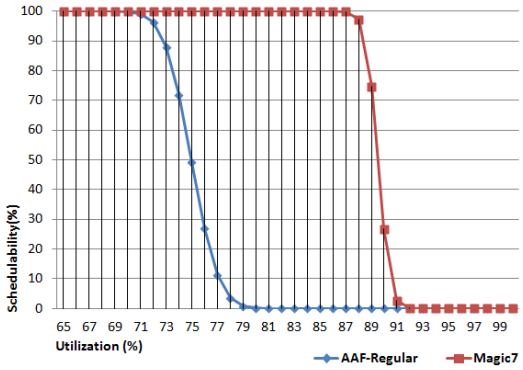


Figure 3.7: 64 Resources, MaxReg=1

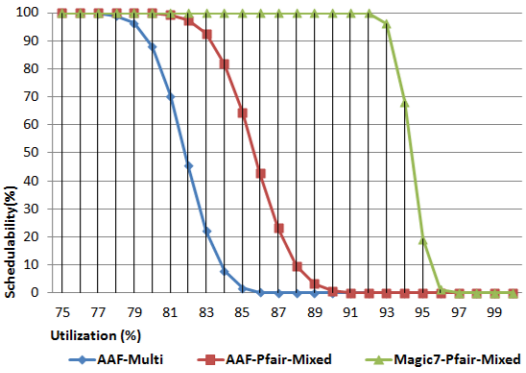


Figure 3.8: 64 Resources, MaxReg=2

partitions, regular and irregular, for assignment to different resources. With it, Problem I is decomposed into two subproblems. One is the problem of scheduling irregular partitions and the other is Problem II.

- We mitigate the resource wasting problem due to the partitioned strategy.
- We demonstrate that the Pfair algorithm can be used to schedule irregular partitions.

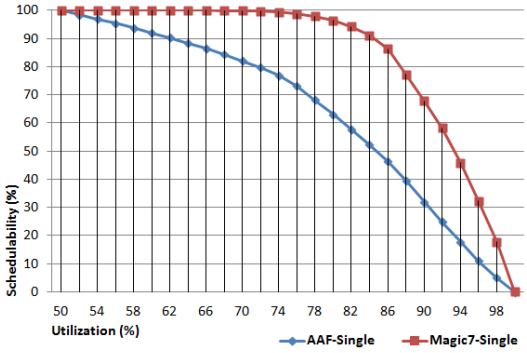


Figure 3.9: Single Resource, MaxReg=2

• We introduce a category of resource partitioning algorithms – Static Approximation Algorithms – for solving Problem II. We prove that the schedulability bound of a feasible Static Approximation Algorithm is at most 0.5, and derive the one with the highest resource utilization (called Magic7) while keeping this maximum schedulability bound.

• Since there is no partition-migration problem on a single-resource platform, our Magic7-enhanced algorithms perform perfectly in this scenario.

There are still many open issues to be investigated. For example, Dynamic Approximation Algorithms could be studied in future. Another, the partition-migration overhead could be taken more into account in the multiresource scenarios.

Chapter 4

MulZ: Noval Partitioned Resource Partitioning

Most Hierarchical Real-time Scheduling (HiRTS) techniques have focused on temporal resource partitions in which time units are periodically distributed. Although such periodic partitions could provide great flexibility for the resource-level scheduling, engineers face significant obstacles when trying to determine the schedulability of real-time tasks running on them. The main reason is that periodic partitions fail to effectively bound the difference between the ideal and the actual resource allocation. To solve this problem, the RRP Model and the Regular Partition is introduced. The Regular Partition is a type of temporal resource partition which is almost evenly distributed. We will show in Chapter 5 that it achieves maximal transparency for task scheduling — some classical real-time scheduling problems on a regular partition can be easily transformed into equivalent problems on a dedicated single resource.

However, the resource partitioning problem for regular partitions is much more complicated than the one for periodic partitions. Based on a practical 2-layer HiRTS platform shown in Figure 1.1, this chapter introduces MulZ which is the first to solve this problem with a partitioned-scheduling strategy. By using a more complicated approximation methodology, our experimental results show that MulZ outperforms the optimal global-scheduling algorithm we introduced in Chapter 3. We also compare the overall performance of the periodic partition and the regular partition. We conclude that the regular partition is a better choice for the integration of real-time applications.

4.1 Single-Resource Scheduling for Regular Partitions

Although there already have been some approximation algorithms for single-resource scheduling of regular partitions, such as AAF [2] and Magic7-Single [13], these algorithms have not been proved *optimal*. We will deeply study this problem by also probing into approximation algorithms in this section. Let us make a naming convention first. We use $\underline{m\text{-resource}}$ to represent a multiresource with m identical resource-units. Specially, $\underline{1\text{-}resource}$ represents a single-resource. Then, we can define the feasibility of an ABS/E-ABS in Def. 4.1.1, where $\{n_i \times w_i : i = 1, 2, ..., k\}_{rp}$ denotes a partition group containing n_i times RP_{w_i}, and $1 \times w_i$ can be simplified to w_i.

Definition 4.1.1 *An ABS/E-ABS B is globally-feasible if and only if* $\forall b_i \in B$ $(i = 1, 2, ..., n)$ *where* $\sum_{i=1}^{n} b_i \leq m$, $\{b_i : i = 1, 2, ..., n\}_{rp}$ *is schedulable on an m-resource via global scheduling.*

The thought behind "B is globally-feasible" is: For a group of regular partitions $\{w_i : i = 1, 2, ..., n\}_{rp}$, we first approximate the weight of each partition at its closest boundary in B using $R_B(w)$. If the sum of these approximated weights does not exceed m, they are always schedulable on an m-resource via global scheduling. On the contrary, if we are able to find a partition group as a counter example that is unschedulable after being approximated by B via global scheduling, we can claim that B is not globally-feasible.

Name	*Definition*	*Globally-Feasible*
$\mathcal{G}_{n,m}$	$\langle \frac{1}{n \cdot m}, \frac{1}{n \cdot m^2}, \frac{1}{n \cdot m^3}, ... \rangle$	yes
$\mathcal{H}_{n,m}$	$\langle \frac{n-1}{n}, \frac{n-2}{n}, ..., \frac{1}{n}, \frac{1}{n \cdot m}, \frac{1}{n \cdot m^2}, \frac{1}{n \cdot m^3}, ... \rangle$	iff $n \in$ RMN (def. 3.3.7)
$\mathcal{Z}_{n,m}$	$(\mathcal{H}_{n,m}, \mathcal{G}_{n,m})$	iff $n \in$ RMN

Table 4.1: Typical ABSes and E-ABSes

Table 4.1 describes some types of ABSes/E-ABSes defined in [13] and their feasibility for global scheduling, and we also list some specific ABSes/E-ABSes as follows, which are widely used in our later discussion. Specially, $\mathcal{Z}_{7,2}$ is the optimal ABS/E-ABS for global scheduling found in [13].

$\mathcal{G}_{1,2} = \langle \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, ... \rangle$

$\mathcal{Z}_{2,2} = \mathcal{Z}_{4,2} = \langle ..., \frac{31}{32}, \frac{15}{16}, \frac{7}{8}, \frac{3}{4}, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, ... \rangle$

$\mathcal{Z}_{3,2} = \langle ..., \frac{23}{24}, \frac{11}{12}, \frac{5}{6}, \frac{2}{3}, \frac{1}{3}, \frac{1}{6}, \frac{1}{12}, \frac{1}{24}, ... \rangle$

$\mathcal{Z}_{5,2} = \langle ..., \frac{39}{40}, \frac{19}{20}, \frac{9}{10}, \frac{4}{5}, \frac{3}{5}, \frac{2}{5}, \frac{1}{5}, \frac{1}{10}, \frac{1}{20}, \frac{1}{40}, ... \rangle$

$\mathcal{Z}_{7,2} = \langle ..., \frac{55}{56}, \frac{27}{28}, \frac{13}{14}, \frac{6}{7}, \frac{5}{7}, \frac{4}{7}, \frac{3}{7}, \frac{2}{7}, \frac{1}{7}, \frac{1}{14}, \frac{1}{28}, \frac{1}{56}, ... \rangle$

Then, we define the ABS/E-ABS feasibility on a single resource as follows. For convenience, we say a regular partition group is *on-1-schedulable* if it is schedulable on a single resource.

Definition 4.1.2 *An ABS/E-ABS B is on-1-feasible if and only if $\forall b_i \in B$ $(i = 1, 2, ..., n)$ where $\sum_{i=1}^{n} b_i \leq 1$, $\{b_i : i = 1, 2, ..., n\}_{rp}$ is on-1-schedulable.*

This definition is very similar to Def. 4.1.1 for *globally-feasible*, but *on-1-feasible* only requires that an ABS/E-ABS always works on a 1-resource. Therefore, "*B is globally-feasible*" is sufficient but not necessary for "*B is on-1-feasible*". There are some ABSes/E-ABSes that are *on-1-feasible* but not *globally-feasible*. $\langle \frac{2}{3}, \frac{1}{2}, \frac{1}{6}, \frac{1}{12}, \frac{1}{24}, ... \rangle$ is such an example. We formally examine it as follows.

Observation 4.1.1 *$\langle \frac{2}{3}, \frac{1}{2}, \frac{1}{6}, \frac{1}{12}, \frac{1}{24}, ... \rangle$ is on-1-feasible.*

Proof. Let B denote the given ABS. Following Def. 4.1.2, we only need to prove that $\forall b_i \in B$ $(b_1 \geq b_2 \geq ... \geq b_n)$ where $\sum_{i=1}^{n} b_i \leq 1$, $G = \{b_i : i = 1, 2, ..., k\}_{rp}$ is on-1-schedulable. CASE 1: $b_1 \leq \frac{1}{6}$. In Table 4.1, we know $\mathcal{G}_{3,2} = \langle \frac{1}{6}, \frac{1}{12}, \frac{1}{24}, ... \rangle$ is globally-feasible, then it is also on-1-feasible. It follows that G is on-1-schedulable in this case. CASE 2: $b_1 = \frac{1}{2}$. If $b_2 = \frac{1}{2}$, it is easy to schedule G; otherwise, we need to examine whether $G' = \{b_i : i = 2, ..., k\}_{rp}$ is schedulable on a $\frac{1}{2}$-weight regular partition, which is equivalent to that $G'' = \{2b_i : i = 2, ..., k\}_{rp}$ is on-1-schedulable, where all

items in G'' belong to $\langle \frac{1}{3}, \frac{1}{6}, \frac{1}{12}, \frac{1}{24}, ... \rangle$. From Table 4.1, $\mathcal{H}_{3,2} = \langle \frac{2}{3}, \frac{1}{3}, \frac{1}{6}, \frac{1}{12}, \frac{1}{24}, ... \rangle$ is on-1-feasible, and it follows that G'' is on-1-schedulable. CASE 3: $b_1 = \frac{2}{3}$. It is obvious that G does not contain $\frac{1}{2}$. Since $\mathcal{H}_{3,2} = \langle \frac{2}{3}, \frac{1}{3}, \frac{1}{6}, \frac{1}{12}, \frac{1}{24}, ... \rangle$ is on-1-feasible, G is on-1-schedulable. □

Observation 4.1.2 $\langle \frac{2}{3}, \frac{1}{2}, \frac{1}{6}, \frac{1}{12}, \frac{1}{24}, ... \rangle$ *is not globally-feasible.*

Proof. We only need to show that partition group $G = \{2 \times \frac{2}{3}, \frac{1}{2}\}_{rp}$ is unschedulable on a 2-resource. Figure 2.2 presents the three forms that $\frac{2}{3}$-weight regular partitions have. CASE 1: If the two $\frac{2}{3}$-weight regular partitions in G is same, without loss of generality, suppose their time-unit sequences are both $T_0(3, 2)$. Obviously, the remaining time units (a couple of $\langle 2, 5, 8, ... \rangle$) cannot produce a $\frac{1}{2}$-weight regular partition, whose time-unit sequence should be $\langle 0, 2, 4, ... \rangle$ or $\langle 1, 3, 5, ... \rangle$. CASE 2: Otherwise, without loss of generality, suppose G contains the first two forms of partitions in Figure 2.2, and the remaining time units $\langle 0, 2, 3, 5, 6, 8, ... \rangle$ also cannot produce a $\frac{1}{2}$-weight regular partition. □

4.1.1 Schedulability Bound on a Single Resource

We figure out the upper limit of the schedulability bound of an on-1-feasible approximation sequence in Theorem 4.1.1 by proving that it is not on-1-feasible if its schedulability bound exceeds 0.5.

Lemma 4.1.1 $\forall$ *on-1-feasible ABS/E-ABS* B *where* $\Upsilon_B > 0.5$, $\exists p \geq 3$, $\frac{1}{p} \in B$.

Proof. presented in Section 4.5. □

Lemma 4.1.2 $\forall p > 1$, *after scheduling* $(p-1)$ *times* $\frac{1}{p}$*-weight regular partitions on a 1-resource, the remaining time units compose another* $\frac{1}{p}$*-weight regular partition.*

Proof. It is true because $T(p, 1, \delta) = \langle \delta \rangle$ for $\delta \in [0, p)$. □

Theorem 4.1.1 $\forall$ *on-1-feasible ABS/E-ABS* B, $\Upsilon_B \leq 0.5$.

Proof. If $\Upsilon_B > 0.5$, by Lemma 4.1.1, $\exists p \geq 3$, $\frac{1}{p} \in B$. And by Theorem 3.3.2, $\exists w \in B, w \in (\frac{1}{2p}, \frac{1}{p})$. Then $\{(p-1) \times \frac{1}{p}, w\}_{rp}$ is not on-1-schedulable because $w \not\prec_{rp} \frac{1}{p}$ (Lemmas 4.1.2, 2.4). This is a counter example against that B is on-1-feasible. It follows that Υ_B cannot be great than 0.5. □

4.1.2 Sub-optimal Approximation on Single Resources

Next, we start to consider the approximation overhead. Based on the shapes shown in Figure 3.5, intuitively, the approximation overhead will decrease if we add more elements into the sequence. So we first define the containment relation between approximation sequences.

Definition 4.1.3 *Suppose A,B are ABSes/E-ABSes,* $A \subseteq B$ *if* $\forall b \in A$, $b \in B$*;* $A \subset B$ *if* $A \subseteq B$ *but* $B \not\subseteq A$.

The next conclusion is obvious from Figure 3.5:

Lemma 4.1.3 *Suppose A,B are ABSes/E-ABSes,* $\mathcal{O}_A < \mathcal{O}_B$ *if* $A \subset B$.

Lemma 4.1.3 shows that a feasible approximation sequence can reach its minimum approximation overhead if we cannot add any element into it without violating its

feasibility. This conclusion leads to the idea of 'Saturated'.

Definition 4.1.4 *An ABS/E-ABS A is saturated if A is on-1-feasible, and $\forall b \notin A$, $A \cup \langle b \rangle$ is not on-1-feasible.*

Furthermore, we say a saturated approximation sequence is sub-optimal if it also reaches the maximal schedulability bound.

Definition 4.1.5 *An ABS/E-ABS A is sub-optimal if A is saturated and $\Upsilon_A = 0.5$.*

We find a group of sub-optimal E-ABSes based on the idea of Regularity Magic Numbers [13]. Reminder that $\mathcal{Z}_{n,2}$ has been defined in Table 4.1.

Theorem 4.1.2 *$\mathcal{Z}_{n,2}$ is sub-optimal if $n \in \{3, 4, 5, 7\}$.*

Proof. It is obvious that $\Upsilon_{\mathcal{Z}_{n,2}} = 0.5$ and $\mathcal{Z}_{n,2}$ is on-1-feasible. We only need to prove that $\mathcal{Z}_{n,2}$ is saturated. Since $\mathcal{Z}_{n,2} = \langle ..., \frac{4n-1}{4n}, \frac{2n-1}{2n}, \frac{n-1}{n}, \frac{n-2}{n}, ..., \frac{1}{n}, \frac{1}{2n}, \frac{1}{4n}, ... \rangle$, $\forall b \notin \mathcal{Z}_{n,2}$, there are four cases:

CASE 1: $b \in (\frac{1}{2^k n}, \frac{1}{2^{k-1} n})$ where $k > 0$.

$\{(2^{k-1}n - 1) \times \frac{1}{2^{k-1}n}, b\}_{rp}$ is not on-1-schedulable because $b \not\prec_{rp} \frac{1}{2^{k-1}n}$ (Lemmas 4.1.2, 2.4).

CASE 2: $b \in (1 - \frac{1}{2^{k-1}n}, 1 - \frac{1}{2^k n})$ where $k > 0$.

$\{b, \frac{1}{2^k n}\}_{rp}$ is not on-1-schedulable because $\frac{1}{2^k n} \not\prec_{rp} (1 - b)$ (Lemmas 4.1.2, 2.4).

CASE 3: $b \in (\frac{k-1}{n}, \frac{k}{n})$ and $b + \frac{k}{n} < 1$, where $1 < k < n$.

$\{1 - \frac{k}{n}, b\}_{rp}$ is on-1-schedulable $\Rightarrow b \prec_{rp} \frac{k}{n}$

$\Rightarrow \exists m > 0,\ b = \frac{m+1}{4m+3}$ and $\frac{k}{n} = \frac{2m+1}{4m+3}$ (Lemma 2.3)

$\Rightarrow m = 1, n = 7, k = 3, b = \frac{2}{7}$.

This result contradicts $b > \frac{k-1}{n}$.

CASE 4: $b \in (\frac{k-1}{n}, \frac{k}{n})$ and $b + \frac{k}{n} > 1$, where $1 < k < n$.

$\{b, 1 - \frac{k}{n}\}_{rp}$ is on-1-schedulable $\Rightarrow (1 - \frac{k}{n}) \prec_{rp} (1 - b)$

$\Rightarrow \exists m > 0,\ \frac{k}{n} = \frac{3m+2}{4m+3}$ and $b = \frac{2m+2}{4m+3}$ (Lemma 2.3)

$\Rightarrow m = 1, n = 7, k = 5, b = \frac{4}{7}$.

This result also contradicts $b > \frac{k-1}{n}$. □

4.2 Partitioned Multiresource Scheduling for Regular Partitions

We have deeply studied the single-resource scheduling problem for regular partitions. A maximum schedulability bound and some sub-optimal E-ABSes have been found. Next, we start to investigate the partitioned multiresource-scheduling problem for regular partitions.

There are two steps in a common partitioned-scheduling algorithm: (1) allocate resource partitions (or real-time tasks) to resource-units; and (2) schedule them on each resource-unit. Step 1 has two concerns when allocating a resource partition: one is which resource-units can contain it; the other is that if multiple resource-units can

do it, which one should be chosen. These two issues correspond to the single-resource scheduling algorithm and the allocation algorithm, respectively.

We adopt the naming convention in [32], where SA-RA denotes the partitioned-scheduling algorithm combining a reasonable allocation algorithm RA and a single-resource scheduling algorithm SA. For example, $\mathcal{Z}_{3,2}$-WF represents the combination of the Worst-Fit-First resource-allocation algorithm and the approximation single-resource scheduling algorithm using $\mathcal{Z}_{3,2}$. Also, Υ_{SA-RA} denotes the Schedulability Bound of SA-RA.

4.2.1 Using a Single Approximation Sequence

There are some particularity when scheduling regular partitions. Since we use an approximation methodology for single-resource scheduling, we can approximate the regular partitions before the allocation step. If the approximation algorithm is based-on one approximation sequence, we only need to guarantee that the total weight allocated to each resource-unit does not exceed 1. We easily conclude that the schedulability bound of such a partitioned algorithm cannot exceed 0.5.

Theorem 4.2.1 $\forall$ *ABS/E-ABS* B, $\Upsilon_{B-RA} \leq \Upsilon_B$.

Proof. Since the weight of each regular partition has to be approximated by B, no matter how these partitions are allocated, Υ_{B-RA} cannot exceed Υ_B. □

Corollary 4.2.1 $\forall$ *ABS/E-ABS* B,$\Upsilon_{B-RA} \leq 0.5$.

We assume that the weight of each regular partition is static, such that we can sort

the regular partitions by their weights before the allocation step. Some allocation algorithms are based on this assumption, such as First Fit Decreasing (FFD) and Best Fit Increasing (BFI). We first notice that $\mathcal{G}_{1,2}$-FFD and $\mathcal{Z}_{2,2}$-FFD reach the maximal schedulability bound shown in Corollary 4.2.1.

Lemma 4.2.1 $\Upsilon_{\mathcal{G}_{1,2}-FFD} = 0.5$.

Proof. From Table 4.1, $\mathcal{G}_{1,2} = \langle \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, ... \rangle$, and $\Upsilon_{\mathcal{G}_{1,2}} = 0.5$. Suppose $\{n_i \times \frac{1}{2^i} : i = 1, 2, 3, ...\}_{rp}$ is the approximated partition set. Obviously, these partitions can be successfully allocated to an m-resource by FFD when $\sum_{i>0} \frac{n_i}{2^i} \leq m$. Therefore, $\mathcal{G}_{1,2}$-FFD always works when the total weight of the original partitions is not greater than 0.5. It follows $\Upsilon_{\mathcal{G}_{1,2}-FFD} \geq 0.5$. By Corollary 4.2.1, $\Upsilon_{\mathcal{G}_{1,2}-FFD} = 0.5$. □

Lemma 4.2.2 $\Upsilon_{\mathcal{Z}_{4,2}-FFD} = 0.5$.

Proof. $\mathcal{Z}_{4,2} = \langle ..., \frac{7}{8}, \frac{3}{4}, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, ... \rangle$. Let's compare the scheduling of $\mathcal{Z}_{4,2}$-FFD and $\mathcal{G}_{1,2}$-FFD. Suppose $\{N_i \times (1 - \frac{1}{2^i}) : i = 2, 3, ...; n_i \times \frac{1}{2^i} : i = 1, 2, 3, ...\}_{rp}$ is the approximated partition set by $\mathcal{Z}_{4,2}$, then $\{(\sum_{i>1} N_i) \times 1;\ n_i \times \frac{1}{2^i} : i = 1, 2, 3, ...\}_{rp}$ is the one by $\mathcal{G}_{1,2}$. Figure 4.1 shows the partition allocations of these two algorithms.

Their allocations of heavy partitions (weight $\geq \frac{1}{2}$) are exactly the same. The difference is between the light-partition allocations. Since $\mathcal{Z}_{4,2}$-FFD leaves some blanks in the heavy part, some light partitions could be assigned into these blanks. Nevertheless, these changes do not negatively impact the schedulability. □

Then we prove that each approximation sequence with the maximal schedulability bound has to contain $\frac{1}{2}$.

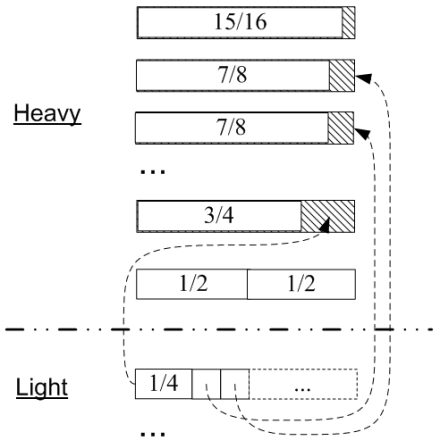


Figure 4.1: $\mathcal{Z}_{4,2}$-FFD and $\mathcal{G}_{1,2}$-FFD

Lemma 4.2.3 *If* $\frac{1}{2} \notin B$, *then* $\forall$ *RA,* $\Upsilon_{B-RA} \leq \max\{0, b : b \in B, b < \frac{1}{2}\}$.

Proof. Let $b_1 = \max\{0, b : b \in B, b < \frac{1}{2}\}$ and $b_2 = \min\{1, b : b \in B, b > \frac{1}{2}\}$. Schedule $\{(m+1) \times (b_1 + \epsilon)\}_{rp}$ on an m-resource with B, where $\epsilon < b_2 - b_1$. This set is approximated to $\{(m+1) \times b_2\}_{rp}$. It is unschedulable for any allocation algorithm because $b_2 > \frac{1}{2}$. It follows $\Upsilon_{B-RA} \leq \frac{(m+1)\cdot(b_1+\epsilon)}{m} \rightarrow b_1$ when $m \rightarrow \infty$ and $\epsilon \rightarrow 0$. Therefore, $\Upsilon_{B-RA} \leq b_1$. □

Corollary 4.2.2 *If* $\frac{1}{2} \notin B$, *then* $\forall RA$, $\Upsilon_{B-RA} < 0.5$.

Theorem 4.2.2 shows that $\mathcal{Z}_{4,2}$ is the only sub-optimal sequence reaching the maximal schedulability bound.

Lemma 4.2.4 *Given an on-1-feasible ABS/E-ABS* B *where* $\Upsilon_B = 0.5$, *if* $\frac{1}{2} \in B$, *then* $\forall k > 0, \frac{1}{2^k} \in B$ *and* $\forall w \in (\frac{1}{2^{k+1}}, \frac{1}{2^k})$, $w \notin B$.

Proof. Use an inductive method. When $k = 1$, $\frac{1}{2} \in B$. Assume $\exists w \in (\frac{1}{4}, \frac{1}{2})$, $w \in B$.

Check the schedulability of $\{\frac{1}{2}, w\}_{rp}$ on a 1-resource. Suppose $RP_{\frac{1}{2}}$ preempts all the time units at even numbers. No matter in which case among $\frac{1}{w} \in (2,3)$, $\frac{1}{w} \in (3,4)$ and $\frac{1}{w} = 3$, there exist two consecutive time units on RP_w whose distance is 3. It follows that $\{\frac{1}{2}, w\}_{rp}$ is not on-1-schedulable because all the time units at even numbers are already preempted by $RP_{\frac{1}{2}}$. Thus, $\forall w \in (\frac{1}{4}, \frac{1}{2})$, $w \notin B$. When $k \geq 1$, by the inductive assumption, $\frac{1}{2^{k-1}} \in B$ and $\forall w \in (\frac{1}{2^k}, \frac{1}{2^{k-1}})$, $w \notin B$. Then $\frac{1}{2^k} \in B$ (otherwise, $\Upsilon_B < 0.5$ by Theorem 3.3.2). Assume $\exists w \in (\frac{1}{2^{k+1}}, \frac{1}{2^k})$, $w \in B$. Let $p = 2^k \geq 4$, then $\{(p-1) \times \frac{1}{p}, w\}_{rp}$ is not on-1-schedulable because $w \not\prec_{rp} \frac{1}{p}$ (Lemmas 4.1.2, 2.4). Therefore, $\forall w \in (\frac{1}{2^{k+1}}, \frac{1}{2^k})$, $w \notin B$. □

Theorem 4.2.2 *If B is a sub-optimal ABS/E-ABS and $\Upsilon_{B-FFD} = 0.5$, then $B = \mathcal{Z}_{4,2}$.*

Proof. By Lemma 4.2.2 and Corollary 4.2.2, we only need to show that $\mathcal{Z}_{4,2}$ is the only sub-optimal ABS/E-ABS containing $\frac{1}{2}$. Suppose B is such a sequence. By Lemma 4.2.4, $\forall k > 0, \frac{1}{2^k} \in B$ and $\forall w \in (\frac{1}{2^{k+1}}, \frac{1}{2^k})$, $w \notin B$. On the other hand, $\forall k > 0$, $\forall w \in (1 - \frac{1}{2^k}, 1 - \frac{1}{2^{k+1}})$, $w \notin B$; otherwise, $\{w, \frac{1}{2^{k+1}}\}_{rp}$ is on-1-schedulable $\Rightarrow \frac{1}{2^{k+1}} \prec_{rp} (1-w)$, which contradicts Lemma 2.3. It follows $B \subseteq \mathcal{Z}_{4,2}$. Since B is saturated, $B = \mathcal{Z}_{4,2}$. □

4.2.2 Using Multiple Approximation Sequences

The performance of a partitioned-scheduling algorithm strongly depends on its approximation overhead. For a given weight, the approximation overhead is not the same on different approximation sequences. For example, $R_{\mathcal{Z}_{4,2}}(0.45) = 0.5$ and

$R_{\mathcal{Z}_{7,2}}(0.45) = 0.57$. This fact inspires us to use multiple approximation sequences in a partitioned-scheduling algorithm. We call it MulZ when we use $\mathcal{Z}_{3,2}$, $\mathcal{Z}_{4,2}$, $\mathcal{Z}_{5,2}$, $\mathcal{Z}_{7,2}$ simultaneously, and present its pseudocode as follows.

---

(0) *resource-units* $R := \{R_j\{factor = 0, rest = 1\} : j \in [1, m]\}$;

(1) *partitions* $P := \{P_j\{weight, res\text{-}unit = 0\} : j \in [1, s]\}$;

---

(2) *bool* ***MulZ_FFD****()*

(3) *sort* P *in non-increasing order;*

(4) ***for*** $j = 1$ *to* s ***do***

(5) $P_j.resource :=$ *MulZ_FFD_Alloc*$(P_j.weight)$;

(6) ***if*** $P_j.resource = 0$ *return false;*

(7) ***od***

(8) *return true;*

---

(9) *int* ***MulZ_FFD_Alloc****(*w*)*

(10) ***for*** $i = 0; n \in \{3, 4, 5, 7\}; i{+}{+}$ ***do***

(11) $A_i = R_{\mathcal{Z}_{n,2}}(w)$;

(12) ***od***

(13) ***for*** $k = 1$ *to 4* ***do***

(14) $r :=$ *the k-th minimum item in array A;*

(15) $f := n$, *where* w *is approximated at* r *by* $\mathcal{Z}_{n,2}$*;*

(16) ***for*** $j = 1$ *to* m ***do***

(17) ***if*** $R_j.factor = f$ *and* $R_j.rest \geq r$ ***do***

(18) $R_j.rest := R_j.rest - r$*;*

(19) *return j;*

(20) ***od***

(21) ***else if*** $R_j.factor = 0$ ***do***

(22) $R_j.factor := f$*;*

(23) $R_j.rest := 1 - r$*;*

(24) *return j;*

(25) ***od***

(26) ***od***

(27) ***od***

(28) *return 0;*

The first two lines define and initialize the data structures. In line (0), a positive

value of "*factor*", n, indicates that a resource-unit is not empty and the partitions on it are approximated by $\mathcal{Z}_{n,2}$. In line (1), a positive value of "*res-unit*", m, indicates that a partition is assigned to the m-th resource-unit. Function *MulZ_FFD* first sorts the resource partitions in non-increasing order, and then calls *MulZ_FFD_Alloc* within a loop to allocate resource for each partition. Lines 10-12 compute the approximated weights of the sub-optimal E-ABSes, and store them as an array. The loop of lines 16–26 checks the availability of each resource-unit one by one for the current partition, where line 14 determines which E-ABS is chosen for approximation in the current iteration; lines 17–20 search available non-empty resource-units using the chosen E-ABS; lines 21–25 assign the current partition to an empty resource-unit if the condition in line 17 fails for all non-empty resource-units (always having lower indexes in R). If there are empty resource-units remaining when starting function *MulZ_FFD_Alloc*, the loop of lines 16–26 must terminate at either line 19 or line 24 when $k = 1$. Therefore, the branch of lines 21–25 is only executed when $k = 1$. It follows that an empty resource-unit always chooses its working E-ABS such that the minimum approximation overhead is achieved for its first assigned partition.

A Partitioning Example of MulZ-FFD: We will show how to partition $G = \{0.65, 0.6, 0.55, 0.5, 0.35, 0.3, 3{\times}0.25\}_{rp}$ on a 4-resource. Let $U(G, n) = \sum_{w \in G} R_{\mathcal{Z}_{n,2}}(w)$. It is easy to check that $\forall n \in \{3, 4, 5, 7\}$, $U(G, n) > 4$. Therefore, any single $\mathcal{Z}_{n,2}$ does not work in this scenario, even if we use a global scheduling strategy without considering migration overhead. However, MulZ-FFD is able to do that. Let's see how the first partition 0.65 is assigned. When $n = 3, 4, 5, 7$, its approximated weight equals $\frac{2}{3}, \frac{3}{4}, \frac{4}{5}, \frac{5}{7}$, respectively. We choose the minimum one $\frac{2}{3}$, and the working E-ABS is

$\mathcal{Z}_{3,2}$. Then we assign this partition to the No. 0 resource-unit because currently all the resource-units are empty. Meanwhile, the working E-ABS on the No. 0 resource-unit is set to $\mathcal{Z}_{3,2}$. The rest can be done in the same manner, and the final result is shown in Table 4.2. The overall resource utilization is 92.5%. Notice that although $\mathcal{Z}_{7,2}$ cannot achieve the minimum overhead for the last remaining 0.25-weight partition, this partition is still assigned to the No. 2 resource-unit because the other three resource-units cannot accommodate it at that moment.

Resource-Unit	E-ABS	Partitions	Approx. Weights
No. 0	$\mathcal{Z}_{3,2}$	0.65, 0.3	$\frac{2}{3}, \frac{1}{3}$
No. 1	$\mathcal{Z}_{5,2}$	0.6, 0.35	$\frac{3}{5}, \frac{2}{5}$
No. 2	$\mathcal{Z}_{7,2}$	0.55, 0.25	$\frac{4}{7}, \frac{2}{7}$
No. 3	$\mathcal{Z}_{4,2}$	0.5, 0.25, 0.25	$\frac{1}{2}, \frac{1}{4}, \frac{1}{4}$

Table 4.2: A Partitioning Example of MulZ-FFD

MulZ-FFD is an intuitive algorithm, which is unable to promise optimal performance theoretically. Nevertheless, our experimental results in the next section show that it has better performance than the best global-scheduling algorithm. Meanwhile, Theorem 4.2.3 shows that the maximal schedulabiltiy bound 0.5 is still kept.

Lemma 4.2.5 *Suppose weights* $w_1, w_2, ..., w_s$ $(w_1 \geq w_2 \geq ... \geq w_s > 0)$ *are already assigned to a resource-unit by MulZ-FFD, where* $\mathcal{Z}_{n,2}$ $(n \in \{3, 4, 5, 7\})$ *is its working E-ABS, and* $\sum_{i=1}^{s} w_i < 0.5$. $\forall w_{s+1} \in (0, w_s]$, *this resource-unit is still able to accommodate* w_{s+1}.

Proof. We only check $\mathcal{Z}_{7,2}$ here. The others can be checked similarly. Let $r_i =$

$R_{\mathcal{Z}_{7,2}}(w_i)$ be the approximated weight of w_i for $i = 1, 2, ..., s{+}1$. Notice $w_i \leq r_i < 2w_i$. Since $\mathcal{Z}_{7,2}$ is sub-optimal, we need to show $U = \sum_{i=1}^{s+1} r_i \leq 1$.

CASE 1: $w_1 \in (\frac{3}{7}, \frac{1}{2})$, $r_1 = \frac{4}{7}$. This case is impossible. When MulZ-FFD assigns w_1 to an empty resource-unit, it should choose $\mathcal{Z}_{4,2}$ as the working E-ABS because $R_{\mathcal{Z}_{4,2}}(w_1) = \frac{1}{2} < r_1$.

CASE 2: $w_1 \in (\frac{2}{7}, \frac{3}{7}]$, $r_1 = \frac{3}{7}$. Similarly, to choose $\mathcal{Z}_{7,2}$ as the working E-ABS when assigning w_1, w_1 must be in $(\frac{2}{5}, \frac{3}{7}]$. If $s = 1$, then $U \leq 2r_1 < 1$; otherwise, $s > 1$ $\Rightarrow \sum_{i=2}^{s} w_i < 0.5 - w_1 < 0.1 \Rightarrow U < r_1 + 2\sum_{i=2}^{s} w_i + 2w_{s+1} \leq \frac{3}{7} + 4\sum_{i=2}^{s} w_i < 1$.

CASE 3: $w_1 \in (\frac{1}{7}, \frac{2}{7}]$, $r_1 = \frac{2}{7}$. To choose $\mathcal{Z}_{7,2}$, w_1 must be in $(\frac{1}{4}, \frac{2}{7}]$. If $s \leq 2$, then $U \leq 3r_1 < 1$; otherwise, $s > 2 \Rightarrow \sum_{i=2}^{s} w_i < 0.5 - w_1 < \frac{1}{4}$:

CASE 3.1: $w_2 \in (\frac{1}{7}, w_1]$, then $\sum_{i=3}^{s} w_i < \frac{1}{4} - \frac{1}{7} = \frac{3}{28}$.

$U < r_1 + r_2 + 2\sum_{i=3}^{s} w_i + 2w_{s+1} \leq \frac{4}{7} + 4\sum_{i=3}^{s} w_i < 1$.

CASE 3.2: $w_2 \leq \frac{1}{7}$.

CASE 3.2.1: $w_s \leq \frac{1}{14}$, then $U < r_1 + 2\sum_{i=2}^{s} w_i + 2w_{s+1} < \frac{2}{7} + \frac{1}{2} + \frac{1}{7} < 1$.

CASE 3.2.2: $w_s > \frac{1}{14}$, then $\frac{1}{7} \geq w_2 \geq w_3 \geq ... \geq w_s > \frac{1}{14}$. Since $\sum_{i=2}^{s} w_i < \frac{1}{4}$, $s \leq 4$. $U = r_1 + \sum_{i=2}^{s} r_i + r_{s+1} \leq r_1 + 4r_2 = \frac{2}{7} + 4 \cdot \frac{1}{7} < 1$.

CASE 4: $w_1 \leq \frac{1}{7}$, then (i) for $i = 1, 2, ..., s+1$, $r_i \in \{\frac{1}{7}, \frac{1}{14}, \frac{1}{28}, \frac{1}{56}, ...\}$; (ii) $r_1 \geq r_2 \geq ... \geq r_s \geq r_{s+1}$. Since $\sum_{i=1}^{s} w_i < 0.5$ and $r_i < 2w_i$, $(1 - \sum_{i=1}^{s} r_i) > 0$. By (i) and (ii), $\forall i \in [1, s]$, r_i is divisible by r_{s+1}. It follows $(1 - \sum_{i=1}^{s} r_i)$ is divisible by r_{s+1}. Therefore, $r_{s+1} \leq 1 - \sum_{i=1}^{s} r_i \Rightarrow U \leq 1$. □

Theorem 4.2.3 $\Upsilon_{MulZ-FFD} = 0.5$.

Proof. (i) Since $\{(m+1) \times (0.5+\epsilon)\}_{rp}$ is unschedulable on an m-resource by MulZ-FFD, $\Upsilon_{MulZ-FFD} \leq \frac{(m+1)\cdot(0.5+\epsilon)}{m} \rightarrow 0.5$ when $m \rightarrow \infty$ and $\epsilon \rightarrow 0$. It follows $\Upsilon_{MulZ-FFD} \leq 0.5$. (ii) Suppose $\{w_1, w_2, ..., w_n : 1 \geq w_1 \geq w_2 \geq ... \geq w_n > 0\}_{rp}$ is unschedulable on an m-resource by MulZ-FFD. Find the proper t where $\{w_i : i = 1, 2, ..., t\}_{rp}$ is schedulable and $\{w_i : i = 1, 2, ..., t+1\}_{rp}$ is unschedulable. If $\sum_{i=1}^{t} w_i < 0.5$, there is a resource-unit whose utilization is less than 0.5. By Lemma 4.2.5, it is able to accommodate a regular partition of weight w_{t+1}. This contradicts that $\{w_i : i = 1, 2, ..., t+1\}_{rp}$ is unschedulable. Therefore, $\sum_{i=1}^{t} w_i \geq 0.5$. It follows that the total weight of any unschedulable partition set is greater than 0.5 and $\Upsilon_{MulZ-FFD} \geq 0.5$. □

4.3 Experimental Results

4.3.1 Regular Partition Scheduling on Mulitresources

Let us briefly explain this part of the simulation experiments. For each weight percentile, we generate 50000 random partition sets. In these sets, the weight of each regular partition is randomly chosen in the interval $\Theta = (low, high)$. Then we simulate different partitioned-scheduling algorithms and count their schedulability rates.

When one approximation sequence is applied, Figure 4.2 shows that $\mathcal{Z}_{7,2}$-FFD

has the highest overall schedulability rate due to its lowest approximation overhead.

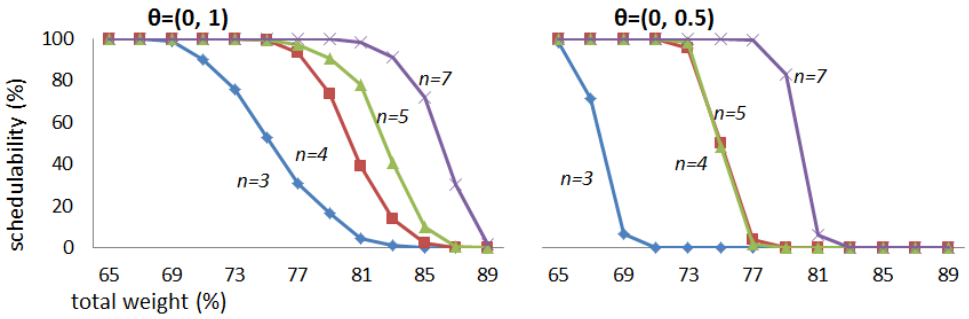


Figure 4.2: Schedulability % of $\mathcal{Z}_{n,2}$-FFD on a 64-resource

Figure 4.3 compares the schedulability rate among MulZ-FFD, $\mathcal{Z}_{7,2}$-FFD, and $\mathcal{Z}_{7,2}$-Global. We have shown that $\mathcal{Z}_{7,2}$-FFD achieves the optimal overall resource utilization with a single E-ABS. Meanwhile, $\mathcal{Z}_{7,2}$-Global (Magic7 [13]) is the optimal global-scheduling algorithm for regular partitions. We ignore its migration overhead because it is hard to estimate a proper value for it. This kind of overhead depends on the physical platform architecture, which is beyond the scope of our work. Meanwhile, this absence will not impact our conclusions.

The simulation results indicate that MulZ-FFD has better performance than the others. First, each chart shows that MulZ-FFD outperforms $\mathcal{Z}_{7,2}$-FFD, which means MulZ-FFD drastically improves the overall resource utilization by using multiple approximation sequences to reduce the approximation overhead. Second, even without considering the migration overhead due to global scheduling, MulZ-FFD performs better or no worse than $\mathcal{Z}_{7,2}$-Global when $\Theta = (0, 1)$, and significantly outperforms $\mathcal{Z}_{7,2}$-Global for light-weight partitions where $\Theta = (0, 0.5)$. Moreover, MulZ-FFD

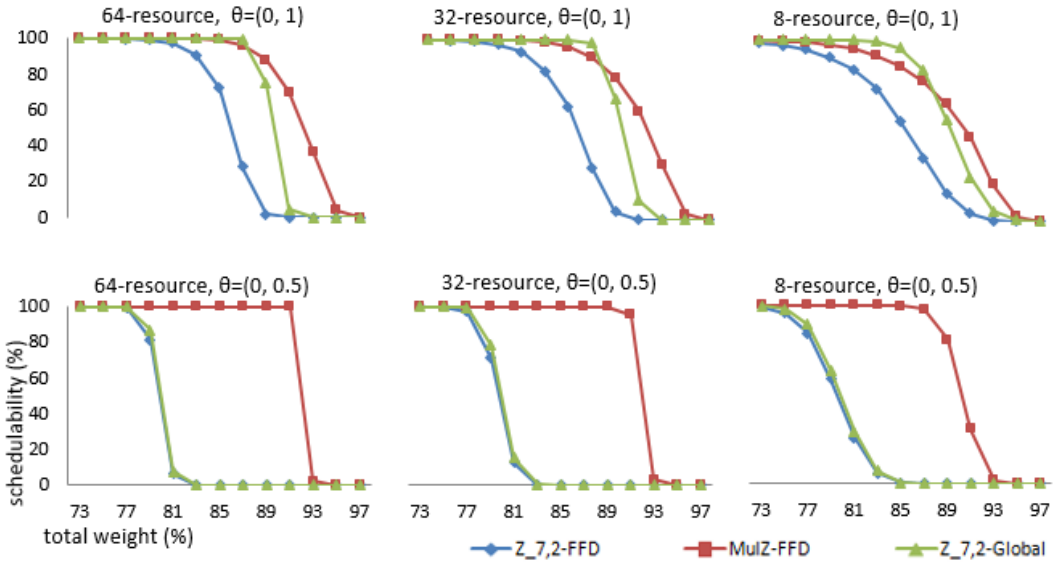


Figure 4.3: MulZ-FFD Greatly Improves Schedulability %

performs even better on middle-to-large multiresource platforms.

4.3.2 Compare the Periodic Model and the Regularity-based Model

Then we compare the overall resource utilization between the Periodic and the Regularity-based Models. As shown in Table 1.2, for task-level scheduling in the Periodic Model, we can only find solutions for the simple periodic-task model. Therefore, our comparison only focuses on this task model, where a periodic task, t_i, is defined as (c_i, p_i). c_i and p_i are t_i's execution time and period, respectively.

At the beginning of the simulation, a size of the computation resource, m, is selected. The main body is a 10000-run loop. In each run, we generate a group of

periodic task sets $\{T_0, T_1, ...T_n : T_i = \{t_{i0}, t_{i1}, ...\}\}$ for each weight percentile r, where the total weight of these task sets is exactly $m \cdot r/100$ and the weight of each task set is randomly chosen in the interval $\Theta = (low, high)$. It follows two major phases. Phase I simulates task scheduling. For each task set T_i, we determine a resource partition P_i which has the exact size to accommodate it with the EDF policy. And Phase II simulates resource partitioning. We determine the schedulability of resource partitions $P_0, P_1, ..., P_n$ on an m-resource. After the 10000-run main loop, we count the schedulability rate.

We implement our simulation in three scenarios: on a single resource, on a multiresource with global scheduling and on a multiresource with partitioned scheduling. In all of them, we apply schedulability tests in [12] and [35] for task scheduling in the Periodic and the Regularity-based Models, respectively. Figure 4.4 shows our experimental results on a single resource. We apply EDF and $\mathcal{Z}_{7,2}$ for resource scheduling in the two models, respectively. The results show that the Regularity-based Model has higher schedulability rate most of the time. Figure 4.5 shows the experimental results on a 64-resource with global scheduling. P-fair and Magic7 are applied for resource scheduling. We find that the Regularity-based Model also outperforms the Periodic Model especially when task sets are light ($\Theta = (0, 0.5)$). Figure 4.6 shows the experimental results on a 64-resource with partitioned scheduling. EDF-FFD and MulZ-FFD are applied for resource scheduling. The results show that the Regularity-based Model outperforms the Periodic Model in both scenarios, no matter the task sets are heavy or light. In general, the Regularity-based Model achieves higher schedulability rate than the Periodic Model, which shows that it also provides

higher overall resource utilization.

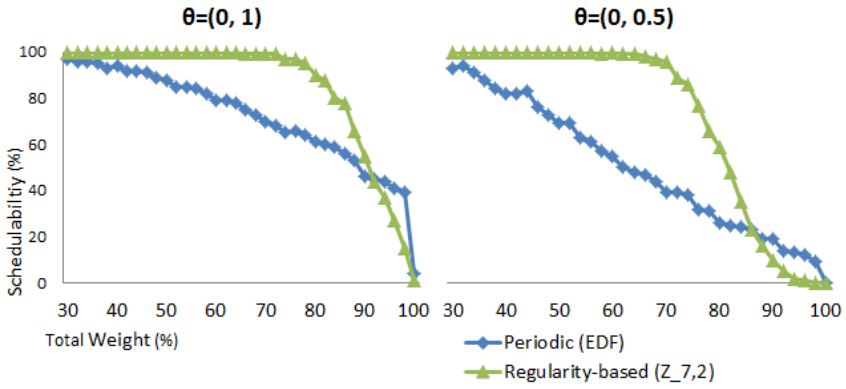


Figure 4.4: Schedulability % on a Single Resource

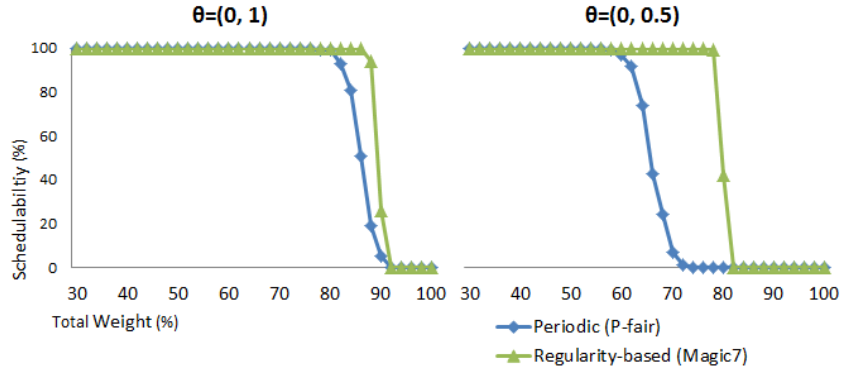


Figure 4.5: Schedulability % on a 64-resource with Global Scheduling

4.4 Contribution

The Periodic Model is the most popular resource model for HiRTS due to its simplicity in resource partitioning. However, it has not solved most classical task

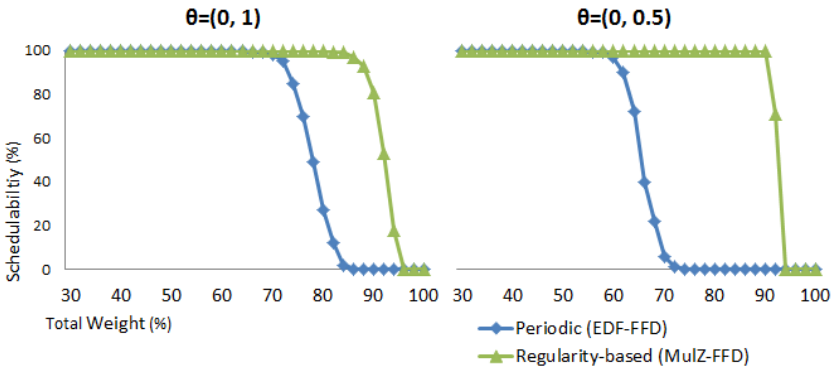


Figure 4.6: Schedulability % on a 64-resource with Partitioned Scheduling

scheduling problems because of the significant blacked-out intervals on its resource partitions. The Regularity-based Model achieves maximal transparency for task scheduling, but its resource-partitioning problem is complicated due to the very strict timing constraint on regular partitions. To alleviate the weaknesses of the Regularity-based Model, we introduces new resource-partitioning techniques for it. After applying these new techniques, our simulation results show that the Regularity-based Model achieves higher overall resource utilization than the Periodic Model. Since the Regularity-based Model can also handle much more task models at the task level, we conclude that the Regularity-based Model is a better choice than the Partitioned Model for the integration of real-time applications.

4.5 Proof of Lemma 4.1.1

Lemma 4.5.1 shows that the time units on a regular partition are always evenly distributed. For convenience, a span denotes two neighboring time units on a regular

partition. Meanwhile, the size of a span is the distance between the two time units it represents.

Lemma 4.5.1 *The size of any span on a regular partition is either* $\lfloor \frac{1}{w} \rfloor$ *or* $\lceil \frac{1}{w} \rceil$*, where* w *is the partition weight. Both types of spans coexist if* $\frac{1}{w}$ *is a fraction.*

Proof. Suppose $w = \frac{q}{p}$ where p, q are co-prime. We only need to check the property on the standard regular partition P with time-unit sequence $T_0(p, q)$. By Def. 2.7, the first time unit preempted on P must be 0; otherwise, the instant regularity at time 1 is less than 0. Notice that $\forall n \in (0, q), \lfloor \frac{n \cdot p}{q} \rfloor \cdot \frac{q}{p} \leq n; (\lfloor \frac{n \cdot p}{q} \rfloor + 1) \cdot \frac{q}{p} > n \Rightarrow \lfloor \frac{n \cdot p}{q} \rfloor \cdot \frac{q}{p} \in (n - \frac{q}{p}, n]$. Consider the second time unit on P. By Lemma 2.1, a preempted time unit increases the instant regularity of P by $(1 - \frac{q}{p})$, and a non-preempted time unit decreases it by $\frac{q}{p}$. To ensure the value of every instant regularity in [0,1) (Def.s 2.7 and 3.3.1), the second preempted time unit must be $\lfloor \frac{p}{q} \rfloor$ because $\lfloor \frac{p}{q} \rfloor \cdot \frac{q}{p} \in (1 - \frac{q}{p}, 1]$ [†]. Similarly, the third time unit must be $\lfloor \frac{2p}{q} \rfloor$; ... the q-th time unit must be $\lfloor \frac{(q-1)p}{q} \rfloor$. Therefore, $T_0(p, q)$ equals $\langle 0, \lfloor \frac{p}{q} \rfloor, \lfloor \frac{2p}{q} \rfloor, ..., \lfloor \frac{(q-1)p}{q} \rfloor \rangle$. $\forall k \in (0, q]$, let $span(k)$ denote the distance between the k-th and $(k+1)$-th time units in $T_0(p, q) \cup \langle p \rangle$, then $span(k) = \lfloor \frac{k \cdot p}{q} \rfloor - \lfloor \frac{(k-1) \cdot p}{q} \rfloor \in [\lfloor \frac{p}{q} \rfloor, \lfloor \frac{p}{q} \rfloor + 1]$. When $\frac{p}{q}$ is a fraction, $\exists k$, $span(k) = \lfloor \frac{p}{q} \rfloor + 1$; otherwise, the total size of all spans in $T_0(p, q) \cup \langle p \rangle$ is $q \cdot \lfloor \frac{p}{q} \rfloor < p$. □

From Lemma 4.5.1, we know that the distance between two neighboring time units on a regular partition is either $\lfloor \frac{1}{w} \rfloor$ or $\lceil \frac{1}{w} \rceil$, but we are still interested in how these

[†]Suppose the second preempted time unit is t. If $t < \lfloor \frac{p}{q} \rfloor$, then $I_P(t + 1) \geq 1$. If $t > \lfloor \frac{p}{q} \rfloor$, then $I_P(t) < 0$. Both cases contradict that $\forall t', I_P(t') \in [0, 1)$.

distances (or spans) are distributed. To show that, we define $T'(p,q)$ as a sequence containing the sizes of these spans in $T_0(p,q)$ in order. For example, $T_0(15,4) = \langle 0,3,7,11\rangle$, then $T'(15,4) = \langle 3,4,4,4\rangle$. Figure 4.7 shows the details. Due to space limitation of the figure, we use letter $'a'$ instead of number $'10'$, and so on.

Definition 4.5.1 $T'(p,q) = \langle t_{i+1} - t_i : 0 \leq i < q\rangle$, *assuming* $T_0(p,q) = \langle t_i : 0 \leq i < q\rangle$ *and* $t_q = p$.

We define $L(p,q)$ as a sequence containing the indexes of those long spans in $T'(p,q)$ in order. For example, $T'(15,4) = \langle 3,4,4,4\rangle \Rightarrow L(15,4) = \langle 1,2,3\rangle$. Since $T_0(p,q)$ has q spans totally where $(p \mod q)$ of them are long spans with size $\lfloor \frac{p}{q} \rfloor + 1$, $|L(p,q)| = p \mod q$.

Definition 4.5.2 $L(p,q) = \langle i : t'_i = \lfloor \frac{p}{q} \rfloor + 1\rangle$, *assuming* $T'(p,q) = \langle t'_i : 0 \leq i < q\rangle$.

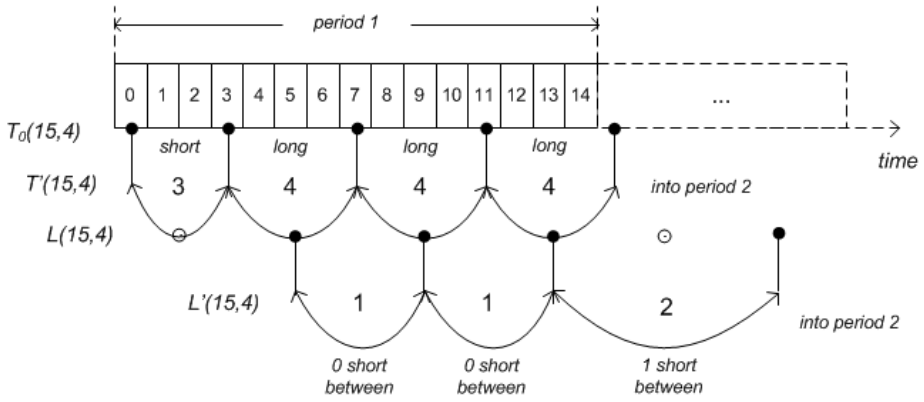


Figure 4.7: Compute $L'(15,4)$ from $T_0(15,4)$

Also, we define $L'(p,q)$ to determine the difference between two neighboring elements in $L(p,q)$. For example, $L'(15,4) = \langle 1,1,2\rangle$. By decreasing the values in $L'(p,q)$ by 1, we directly deduce the number of short spans between two neighboring

long spans. Figure 4.7 shows the computation from $T_0(15,4)$ to $L'(15,4)$.

Definition 4.5.3 $L'(p,q) = \langle l_{i+1} - l_i : 0 \leq i < r \rangle$, *assuming* $L(p,q) = \langle l_i : 0 \leq i < r \rangle$ *and* $l_r = q + l_0$, *where* $r = p \mod q$.

Lemma 4.5.2 shows that not only $T_0(p,q)$, but also $L(p,q)$ are evenly distributed.

Lemma 4.5.2 $\forall p, q$ *where* $p > q > 1$ *and p,q are co-prime,* $\max\{l' : l' \in L'(p,q)\} - \min\{l' : l' \in L'(p,q)\} \leq 1$.

Proof. Let $d = \lfloor \frac{p}{q} \rfloor \geq 1$.

(I) First, let's study the case when all the elements of $L'(p,q)$ are equal to n. $n = 1$ is impossible; otherwise, all the elements of $T'(p,q)$ are equal to $(\lfloor \frac{p}{q} \rfloor + 1)$. Then $n > 1 \Rightarrow |L'(p,q)| = 1$; otherwise, $T'(p,q)$ can be divided into several identical segments, which contradicts the fact that p, q are co-prime (For example, if $L'(p,q) = \langle 2, 2 \rangle$, $T'(p,q) = \langle d, d+1, d, d+1 \rangle \Rightarrow p = 4d+2, q = 4$). Therefore, there is only one element of $(d+1)$ in $T'(p,q)$, together with $(n-1)$ elements of d (Def.s 4.1.4, 4.1.5). Then $q = |T'(p,q)| = n$; $p = nd+1$ (Def. 4.1.3) $\Rightarrow \frac{p}{q} = d + \frac{1}{n}$. Vice versa, if $\frac{p}{q} = d + \frac{1}{n}$ where $n > 1$ and p, q are co-prime, then $p = nd+1, q = n \Rightarrow T_0(p,q) = \langle 0, d, 2d, ..., (n-1)d \rangle$ (Def.s 2.7 and 3.3.1) $\Rightarrow T'(p,q) = \langle d, d, d, ..., d+1 \rangle$ (Def. 4.1.3) $\Rightarrow L(p,q) = \langle n-1 \rangle$ (Def. 4.1.4) $\Rightarrow L'(p,q) = \langle n \rangle$ (Def. 4.1.5).

(II) If there are elements with different values in $L'(p,q)$, $\exists n > 0$, $\frac{p}{q} \in (d + \frac{1}{n+1}, d + \frac{1}{n}) \Rightarrow \frac{q}{p} \in (\frac{n}{nd+1}, \frac{n+1}{nd+d+1})$. Assume $n+k \in L'(p,q)$ where $k \geq 2$. Consider the changes of the supply regularity in $T_0(p,q)$. A short span increases the supply regularity by $(1 - \frac{q}{p} \cdot d)$. With the following preempted time unit, $(n+k-1)$ straight short spans

increase the supply regularity by: $(n+k-1)(1-d\cdot\frac{q}{p})+(1-\frac{q}{p}) \geq (n+1)(1-d\cdot\frac{q}{p})+(1-\frac{q}{p})$ $= n+2-(nd+d+1)\cdot\frac{q}{p} > 1$. It contradicts the definition of regular partition (Def. 2.7). Therefore, $\underline{\forall k \geq 2,\ n+k \notin L'(p,q)}$. Similarly, if $n-k \in L'(p,q)$ where $k \in [1,n)$, excluding the first preempted time unit, two neighboring long spans and the $(n-k-1)$ straight short spans between decrease the supply regularity by: $d\cdot\frac{q}{p}-(n-k-1)(1-d\cdot\frac{q}{p})-(1-\frac{q}{p})+d\cdot\frac{q}{p} \geq 2d\cdot\frac{q}{p}-(n-2)(1-d\cdot\frac{q}{p})-(1-\frac{q}{p}) =$ $(nd+1)\cdot\frac{q}{p}-(n-1) > 1$. It also contradicts the definition of regular partition. Therefore, $\underline{\forall k \in [1,n),\ n-k \notin L'(p,q)}$. The only two possible values in $L'(p,q)$ are n and $n+1$. □

Then we check the following case. $\forall p,q$ and $p > q$, the time-unit sequences $T(p,q,\delta)$ $(\delta = 0,1,...,\lfloor\frac{p}{q}\rfloor-1)$ can be accommodated on a 1-resource without conflict, and each long span of $T_0(p,q)$ leaves its last time unit unused in the end. In Figure 3.5, time units labeled by $'6',' a'$ and $'e'$ are left out from $T_0(15,4), T(15,4,1)$ and $T(15,4,2)$. Let $D(p,q)$ denote these remaining time units. Obviously, $|D(p,q)| =$ $p-q\cdot\lfloor\frac{p}{q}\rfloor = p \mod q$.

Definition 4.5.4 $D(p,q) = \langle 0,1,...,p-1\rangle - \bigcup_{\delta=0}^{\lfloor\frac{p}{q}\rfloor-1} T(p,q,\delta)$.

Also, we define $D'(p,q)$ to show the distances between the neighboring time units in $D(p,q)$. For example, $D(15,4) = \langle 6,10,14\rangle$, $D'(15,4) = \langle 4,4,7\rangle$.

Definition 4.5.5 *$D'(p,q) = \langle t_{i+1}-t_i : 0 \leq i < r\rangle$, assuming $D(p,q) = \langle t_i : 0 \leq i <$ $r\rangle$ and $t_r = t_0 + p$, where $r = p \mod q$.*

Lemma C2 shows how to compute $D'(p,q)$ from $L'(p,q)$. Notice that they both have the size of $(p \mod q)$.

Lemma 4.5.3 *Suppose* $L'(p,q) = \langle l'_i : 0 \le i < r \rangle$ *and* $D'(p,q) = \langle d'_i : 0 \le i < r \rangle$ *where* $r = p \mod q$*, then* $d'_i = l'_i \cdot \lfloor \frac{p}{q} \rfloor + 1$ *for* $0 \le i < r$.

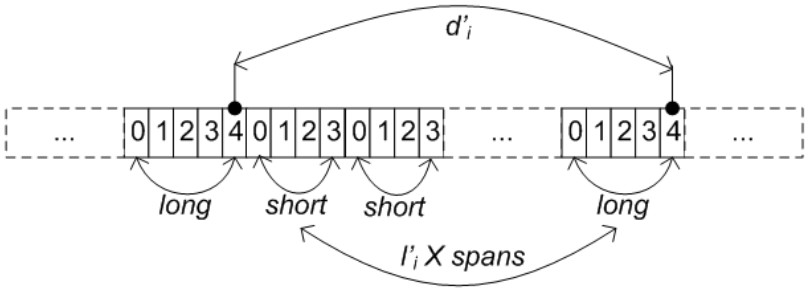


Figure 4.8: Compute $D'(p,q)$ from $L'(p,q)$

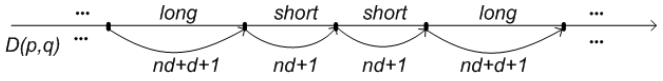
Proof. As shown in Figure 4.8, $D(p,q)$ is composed of the last time units of the long spans (labeled by $'4'$ in the example). Since $(l'_i - 1)$ is the number of short spans between two neighboring long spans, $d'_i = (l'_i - 1) \cdot \lfloor \frac{p}{q} \rfloor + (\lfloor \frac{p}{q} \rfloor + 1) = l'_i \cdot \lfloor \frac{p}{q} \rfloor + 1$ for $0 \le i < r$. □

Lemma 4.5.4 *Suppose* $\frac{q}{p} < \frac{1}{2}$ *where* p, q *are co-prime, and* $r = (p \mod q) > 0$*. If* $\{\lfloor \frac{p}{q} \rfloor \times \frac{q}{p}, w\}_{rp}$ *is on-1-schedulable and* $w \in (\frac{r}{2p}, \frac{r}{p})$*, then* $p \cdot w$ *is an integer.*

Proof. Let $d = \lfloor \frac{p}{q} \rfloor$, then $d \ge 2$. By Lemma 4.5.2, there are two cases for the values in $L'(p,q)$.

CASE 1: $L'(p,q)$ only contains n. We have examined this case in the proof part (I) of Lemma 4.5.2, where we have proved $|L'(p,q)| = 1$. It follows that $|D(p,q)| = 1$ (Lemma 4.5.3), and $D(p,q)$ is the time-unit sequence of a $\frac{1}{p}$-weight regular partition. Hence, $\{\lfloor \frac{p}{q} \rfloor \times \frac{q}{p}, w\}_{rp}$ is <u>not on-1-schedulable</u> because $w \not\prec_{rp} \frac{1}{p}$ (Lemma 2.4).

CASE 2: $L'(p,q)$ contains both n and $n+1$ where $n > 0$. By Lemma 4.5.3, $D'(p,q)$ contains both $nd+1$ and $nd+d+1$, as shown in the next figure. Let $short = nd+1$ and $long = nd+d+1$, then $long > short+1$ and $long < 2 \times short$. Since



$\{\lfloor \frac{p}{q} \rfloor \times \frac{q}{p}, w\}_{rp}$ is schedulable, suppose P_w is the w-weight regular partition, then the time-unit sequence on P_w is a part of $D(p,q)$ within their hyper period. And because $w \in (\frac{r}{2p}, \frac{r}{p})$, P_w contains more than a half of time units in $D(p,q)$. Let $d' = \lfloor \frac{1}{w} \rfloor$, and the size of each span on P_w is either d' or $d'+1$. Then, $d'+1 \geq long$ * and $d' < 2 \times long$ † $\Rightarrow d' \in (short, 2 \times long)$.

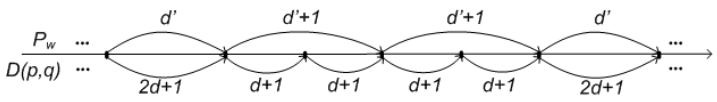
CASE 2.1: $\frac{1}{w}$ is an integer. The time-unit sequence of P_w is an arithmetic one and the size of each span is d' (Lemma 4.5.1). Since $d' \in (short, 2 \times long)$ and $2 \times short > long$, the only possible values of d' are $long, 2 \times short$ and $long + short$. (i) If $d' = long$, P_w cannot go over any short span of $D(p,q)$ because $short < d' < 2 \times short$. (ii) If $d' = 2 \times short$, P_w cannot go over any long span of $D(p,q)$ because $long < d' < long + short$. (iii) If $d' = long + short$, P_w contains exactly a half of the time units in $D(p,q)$ because it selects time units from $D(p,q)$ alternately. It follows $w = \frac{r}{2p}$, which contradicts the assumption $w \in (\frac{r}{2p}, \frac{r}{p})$.

CASE 2.2: $\frac{1}{w}$ is a fraction. There are two sizes of spans on P_w: d', $d'+1$. Since $d' \in (short, 2 \times long)$, the only possible values of d' and $d'+1$ are $long, 2 \times short, long+$

*If $d'+1 < long$, P_w cannot go over any long span of $D(p,q)$.

†Because P_w contains more than a half of the time units in $D(p,q)$, $d' \leq average\_span\_size\_of\_P_w < 2 \times average\_span\_size\_of\_D(p,q) < 2 \times long$.

short, and $2 \times long$, where $long < 2 \times short < long + short < 2 \times long \Rightarrow long + 1 = 2 \times short$ OR $2 \times short + 1 = long + short$ OR $long + short + 1 = 2 \times long \Rightarrow long = 2 \times short - 1$ (because $long > short + 1$) $\Rightarrow n = 1$ (apply $short = nd + 1$ and $long = nd + d + 1$). Therefore, $short = d + 1$, $long = 2d + 1$, $d' = long$ and $d' + 1 = 2 \times short$. Also, the number of the short spans between each two neighboring long spans on $D(p, q)$ must be even. As shown in the next figure, P_w and $D(p, q)$ always keep synchronized — a short span of P_w corresponds to a long span of $D(p, q)$, and a long span of P_w corresponds to two short spans of $D(p, q)$. Since p is a period of $D(p, q)$, p is also a period of P_w. It follows $p \cdot w$ is an integer. □



Lemma 4.1.1 $\forall$ *on-1-feasible ABS/E-ABS* B *where* $\Upsilon_B > 0.5$, $\exists p \geq 3$, $\frac{1}{p} \in B$.

Proof. From Def.s 3.3.1, 3.3.9, $\exists \frac{q_0}{p_0} \in B$, where $\frac{q_0}{p_0} < \frac{1}{2}$ and p_0, q_0 are co-prime. If $q_0 = 1$, just let $p = p_0$; otherwise, let $r_0 = p_0 \mod q_0$, then $r_0 > 0$. Since $\Upsilon_B > 0.5$, by Theorem 3.3.2, $\exists w \in B, w \in (\frac{r_0}{2p_0}, \frac{r_0}{p_0})$. Since B is on-1-feasible, $\{\lfloor \frac{p_0}{q_0} \rfloor \times \frac{q_0}{p_0}, w\}_{rp}$ is on-1-schedulable. By Lemma 4.5.4, $p_0 \cdot w$ is an integer. Suppose $w = \frac{q_1}{p_1}$ where p_1, q_1 are co-prime. Then $p_0 \cdot \frac{q_1}{p_1}$ is an integer $\Rightarrow p_0$ is divisible by p_1. Therefore, $\frac{q_1}{p_1} < \frac{r_0}{p_0} < \frac{q_0}{p_0} \Rightarrow q_1 < \frac{p_1}{p_0} \cdot q_0 \leq q_0$. Repeat the same argument from p_1 and q_1 ..., and eventually, we can find p_n, q_n such that $\frac{q_n}{p_n} \in B$ and $q_n = 1$. Meanwhile, $\frac{q_n}{p_n} < \frac{q_0}{p_0} < 0.5 \Rightarrow \frac{1}{p_n} < 0.5 \Rightarrow p_n \geq 3$. Let $p = p_n$. □

Chapter 5

Transparent Task Scheduling

[†]In this chapter, we study task scheduling on regular partitions to show that they have the transparency property in comprehensive cases. For the scheduling policies contained in a specific category (including the most popular EDF and DM [8]), we can always transform a scheduling problem on a regular partition into an equivalent problem on a dedicated single resource. We extend the study of real-time task scheduling on temporal resource partitions in two dimensions. On the one hand, our method is able to deal with different types of real-time tasks such as periodic tasks, sporadic tasks, and aperiodic tasks. On the other hand, unlike other results currently known in the literature which only consider one scheduling policy at a time, our method works on a category of policies. Meanwhile, it is the first to

[†]The content of this chapter has been published.

allow existing single-resource scheduling algorithms for task scheduling on resource partitions. This is a big convenience for building a HiRTS system because we do not have to develop new task-scheduling techniques.

To achieve the transformation, we introduce coarse-grain scheduling as an intermediate state between the regular-partition task scheduling and the single-resource task scheduling. As shown in Figure 5.1, we perform three major steps to complete the argument of our transformation method. Step ① investigates the relation between a common scheduling problem and a coarse-grain scheduling problem both on a single-resource. Step ② discusses the relation between a coarse-grain scheduling problem on a single-resource and a scheduling problem on a regular-partition assignment. Step ③ eliminates the intermediate state and connects a scheduling problem on a regular partition and one on a single resource. Our final goal is to transform a scheduling problem on a regular partition into one on a single resource that has the same schedulability.

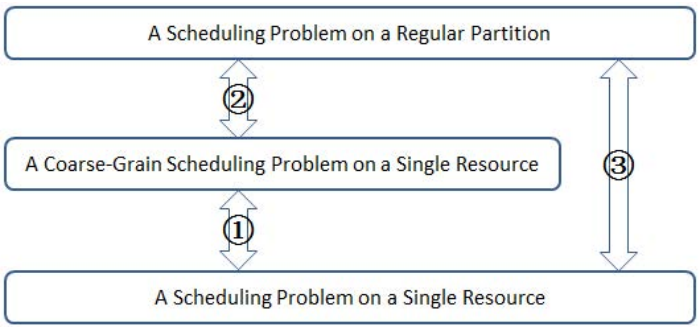


Figure 5.1: Our Transformation Strategy

5.1 Existing Work

Real-time task scheduling has been extensively studied since Liu and Layland's seminal work [8] on the Earliest Deadline First (EDF) and Rate Monotonic (RM) algorithms. We describe a few representative papers here. Lehoczky et al. [19] provides an exact schedulability test for RM. Jeffay et al. [21] describes non-preemptive scheduling of periodic and sporadic tasks. Harbour et al. [22] and Davis and Burns [23] derive response-time bounds for fixed-priority real-time systems. Lauzac et al. [24], Bini et al. [25], and Davis et al. [26] introduce more efficient RM analysis. Lu et al. [27] present a precision-tunable response time upper bound algorithm for fixed-priority preemptive real-time systems. Ras and Cheng [28] and Belwal and Cheng [29] provide response time analysis of abort-and-restart tasks in functional reactive systems. Lin et al. [30] investigate energy-aware scheduling of real-time tasks. More recently, Lin et al. [31] present fault-tolerant scheduling techniques for mixed-criticality real-time tasks. However, all these results consider only single-layer scheduling of tasks whereas our current paper targets hierarchical real-time systems.

We list the existing HiRTS models and their partition definitions in Table 5.1. Since our main focus is to show the transparency property of the regular partition in the RRP Model, we would like to first argue why currently only this kind of partition is able to provide this property. Intuitively, a resource partition could provide perfect transparency property for task scheduling when its resource allocation is even at everywhere. For example, for a partition of weight $\frac{1}{3}$, if it exactly preempts $\frac{1}{3}$ computation resource in any time interval, it could provide perfect transparency.

However, this is just an ideal allocation state. No practical algorithm is able to achieve it because time intervals could be infinitely cut apart. Therefore, no matter what kind of resource-allocation algorithm is applied to a resource partition, there exists a deviation between the ideal and the actual resource allocation.

A resource partition in the Periodic Model [12] exactly obtains c computation time units in each period p as shown in Figure 5.2, and there is no other restriction on the resource allocation. Therefore, there could be some blacked-out intervals without any computation time. The maximal length of such an interval is $2(p-c)$. The EDP Model [14] introduced one additional parameter d to restrict that only the first d time units can get computation time in each period. However, the maximal length of a blacked-out interval is still $(p+d-2\cdot c)$. Since these two models (Periodic and EDP) do not effectively bound the deviation between the ideal and the actual resource allocation, their resource partitions cannot provide the transparency property for task scheduling. New utilization bounds and schedulability tests have to be developed.

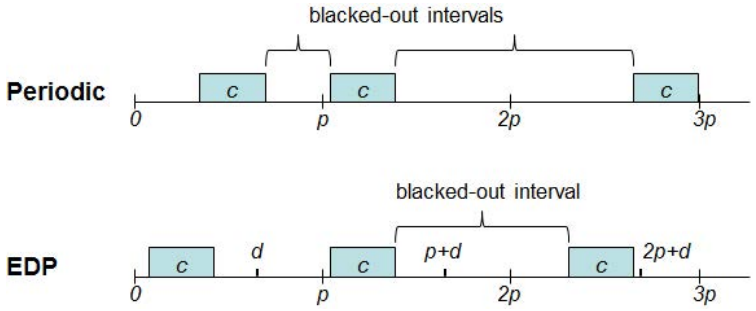


Figure 5.2: Blacked-out Intervals

On the other hand, the Bounded-delay Model [2] and the RRP Model [1, 7] are designed to bound the deviation between the ideal and the actual resource allocation.

Their only difference is that the RRP Model has one more constraint that there is an impartible time unit. Since time intervals could be infinitely cut apart in the Bounded-delay Model, it cannot find a minimum bound for that deviation range. Therefore, its resource partitions cannot provide the transparency property either. Only the regular partition in the RRP Model is claimed to provide transparency in two specific cases shown in Table 5.1, where the regular partition minimizes the deviation range between the ideal and the actual resource allocation.

Table 5.1 summarizes the limitations of current HiRTS techniques. First, only one restricted task model is considered for each HiRTS model. For example, for the Periodic, Bounded-Delay, and RRP Models, the current work has only investigated a special case of the periodic task model – all tasks are synchronous with implicit deadlines. Second, these methodologies study scheduling policies one by one. This strategy greatly increases the research workload of task scheduling for HiRTS. At last, regular partitions (regularity = 1) in the RRP model could provide transparency. However, this is true only in some specific cases – there are strict restrictions on the task model (synchronous, implicit-deadline and periodic) and scheduling policies (EDF or DM).

Our work in this chapter is based on a two-layer HiRTS system as shown in Figure 1.1. We use the definition of resource partition in the RRP Model and focus on regular partitions. The RRP Model originates by Shigero et al. [7], which shows that the utilization bound of EDF remains unchanged on regular partitions for synchronous periodic tasks with implicit deadlines. Then Mok and Feng [1, 3] prove that the utilization bound of DM remains unchanged in the same scenario. To the

	Partition Definition	Task Models	Task Scheduling on Partitions
Periodic [12]	period, weight	periodic (limited)	new utilization bounds: EDF & DM [12]
EDP [14]	period, weight, deadline	sporadic (limited)	new schedulability tests: EDF & DM [14]
Bounded-Delay[2]	weight, jitter	periodic (limited)	new utilization bounds: EDF & DM [2]
RRP [7, 1]	weight, regularity	periodic (limited)	utilization bounds unchanged on regular partitions (regularity = 1): EDF [7]; DM [1]

Table 5.1: The State of the Art of Task Scheduling in HiRTS

best of our knowledge, no existing result on the RRP Model has considered the task scheduling problem based on general real-time tasks as described in this work.

5.2 Prerequisite Knowledge

5.2.1 Job and Priority

Unlike the traditional research on the task scheduling that focuses on the periodic task model, we study how to schedule a set of real-time jobs. As shown in Def. 5.2.1, a job has only one single request with an execution time, a release time, and a deadline. The methodology based on scheduling a sequence of jobs is easily applied to scheduling different types of real-time tasks.

Definition 5.2.1 *$J = (c, r, d)$ denotes a job, where c, r, and d are its execution*

time, release time, and deadline, respectively. For convenience, we use $C(J)$, $R(J)$ and $D(J)$ to denote J's execution time, release time, and deadline, respectively.

We consider preemptive scheduling policies in our work. We regard a preemptive policy as *P-stable* if each job has an invariant as its priority that never changes with time. For example, EDF is P-stable and the priority of a job is its deadline, whereas Round Robin is not P-stable. Once a policy is P-stable, we can define the priority order of a job sequence for this policy as follows.

Definition 5.2.2 *The **priority order** of a job sequence S for a P-stable policy Γ is a non-strict total order $\preceq_p$ on $[0, |S|)$ such that $i \prec_p j$ iff $S|_i$ has a higher priority than $S|_j$ with Γ; $i =_p j$ iff $S|_i$ has the same priority as $S|_j$ with Γ.*

A job is regarded as active at a time instant if it has been released but not finished. A big concern in a policy is how to break ties when multiple active jobs have equal priority. A tie-break order in Def. 5.2.3 is able to resolve this situation. Once the priority order and the tie-break order are determined, we can obtain a scheduling order on a job sequence as in Def. 5.2.4, which indicates which active job will preempt the current time unit at a time instant.

Definition 5.2.3 *Suppose Γ is a P-stable policy, and $\preceq_p$ is the priority order of a job sequence S for Γ. A **tie-break order** of S for Γ is a strict partial order $\prec_t$ on $[0, |S|)$ that is well-defined between i and j iff $i =_p j$, where $i \prec_t j$ indicates that $S|_i$ always precedes $S|_j$ when they are both active.*

Definition 5.2.4 *Suppose Γ is a P-stable policy; $\preceq_p$ and $\prec_t$ are the priority order and a tie-break order of a job sequence S for Γ, respectively. The **scheduling order***

of S for Γ and $\prec_t$ is a strict total order $\prec$ on $[0, |S|)$ such that $i \prec j$ iff $i \prec_p j$, OR $i =_p j$ and $i \prec_t j$.

5.2.2 Resource and Job System

For convenience, we regard a single resource as a special case of a regular-partition assignment whose period is 1, and use $\mathcal{A}^{(1)}$ to represent it. To describe a scheduling problem, we define a regular job system as follows.

Definition 5.2.5 *$\Pi = (R, S, \Gamma, \prec_t)$ denotes a **regular job system**, where R is a regular-partition assignment; S is a sequence of independent jobs; Γ is a preemptive hard-real-time policy; and $\prec_t$ is a tie-break order on S for Γ.*

Definition 5.2.6 *$F(\Pi, i)$ denotes the finish time of $S|_i$ in $\Pi = (R, S, \Gamma, \prec_t)$. If $S|_i$ can never finish, $F(\Pi, i) = \infty$.*

Definition 5.2.7 *$\Pi = (R, S, \Gamma, \prec_t)$ is feasible iff $\forall i \in [0, |S|)$, $F(\Pi, i) \leq D(S|_i)$.*

5.3 Coarse-grain Scheduling

Now, we start to present our transformation strategy. Following the three steps in Figure 5.1, we show some more details pertaining to the scheduling problems in Figure 5.3. Problem I checks the schedulability of a job sequence S under a scheduling policy Γ with a tie break order $\prec_t$ on a regular partition of weight $\frac{q}{p}$. Problem II checks the schedulability of a job sequence S' under the same scheduling policy Γ with the same tie break order $\prec_t$ on a single resource, where S' is obtained from

S with a rule introduced later (job scaling). Our target is to prove that these two problems are equivalent. Therefore, we introduce Problem III as a bridge between them. Problem III checks the schedulability of S' under Γ' (the coarse-grain version of Γ) with the same tie break order $\prec_t$ on a single resource. In this section, we introduce coarse-grain scheduling first and then perform Step ① to connect Problem II and Problem III.

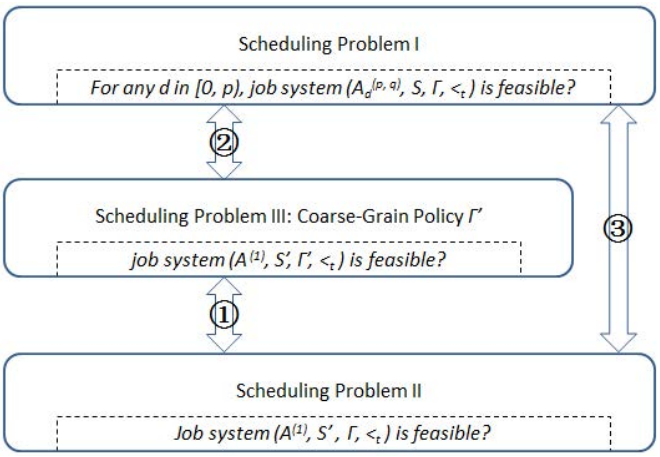


Figure 5.3: The Coarse-grain Scheduling Works as a Bridge

5.3.1 Introducing Coarse-grain Scheduling

The coarse-grain version of a scheduling policy assigns time units to jobs with a fixed frame size p (the first parameter). Meanwhile, jobs are allowed to start at most d (the second parameter) time units earlier. $\forall k \geq 0$, "time frame k" denotes time interval $[k \cdot p, (k+1) \cdot p)$ in the coarse-grain scheduling scenario.

Definition 5.3.1 *For a preemptive scheduling policy Γ applied on a dedicated single*

resource, $\Gamma^{(p,d)}$ $(p > d \geq 0)$ *denotes its coarse-grain version, with which all preemptions occur at time instants of* p*'s multiples, and each job is allowed to start execution at most* d *time units prior to its release time.*

Figure 5.4 shows an example of the coarse-grain EDF with frame size 4. Notice that with EDF$^{(4,1)}$, J_2 starts execution at time 4 though its release time is 5. Another notable fact is that coarse-grain policies could cause unnecessary deadline misses due to their coarse-grain assignment mechanism. As shown in Figure 5.4, all jobs meet their deadlines with EDF, while J_2 and J_3 miss their deadlines with EDF$^{(4,0)}$. In Section V, we will show that this does not matter for our transformation method, and a coarse-grain scheduling problem on a single resource perfectly matches a scheduling problem on a regular partition. Also, for the purpose of transformation, we only need to consider harmonic job sequences (Def. 4.2) in the coarse-grain scheduling scenario.

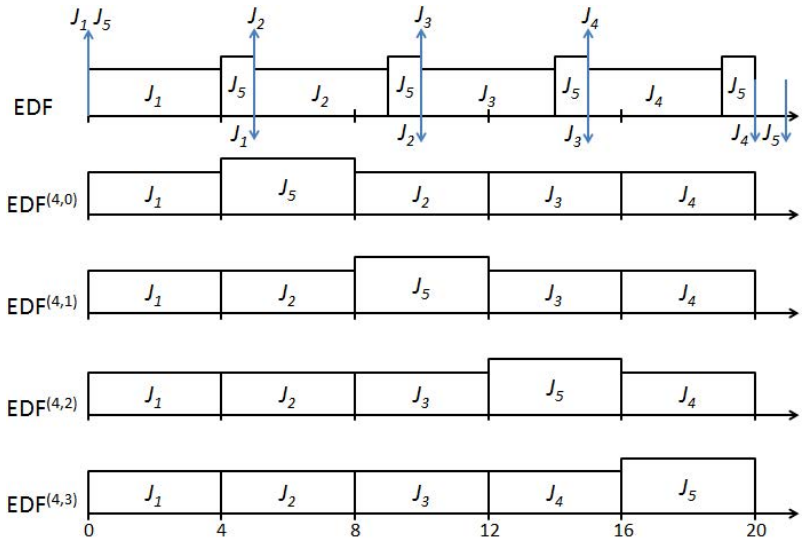


Figure 5.4: Coarse-grain EDF $\{J_1 = (4, 0, 5), J_2 = (4, 5, 10), J_3 = (4, 10, 15), J_4 = (4, 15, 20), J_5 = (4, 0, 21)\}$

Definition 5.3.2 *A job sequence is p-harmonic if the execution time of each job is a multiple of p.*

5.3.2 Connect Coarse-grain Scheduling to Common Single-Resource Scheduling

There is another interesting observation from Figure 5.4 – each time frame in the $\text{EDF}^{(4,d)}$ scenario is assigned to a job that occupies the d-th time unit within this time frame in the EDF scenario. For example, time units 0, 4, 8, 12, 16 in the EDF scenario are assigned to J_1, J_5, J_2, J_3, J_4, respectively. Meanwhile, the first five time frames in the $\text{EDF}^{(4,0)}$ scenario are also assigned to J_1, J_5, J_2, J_3, J_4, respectively. We show that this is a general rule in Lemma 5.3.2, which matches the results of a common scheduling $\Pi = (\mathcal{A}^{(1)}, S, \Gamma, \prec_t)$ and a coarse-grain scheduling $\Pi' = (\mathcal{A}^{(1)}, S, \Gamma^{(p,d)}, \prec_t)$. A coloring method is applied on the single resource used for the common scheduling, which totally uses p colors $C_0, C_1, ..., C_{p-1}$ and paints any time unit t with color $C_{(t \mod p)}$.

Lemma 5.3.1 *Given a regular job system $\Pi = (\mathcal{A}^{(1)}, S, \Gamma, \prec_t)$ where S is p-harmonic and Γ is P-stable, $\forall i \in [0, |S|)$ where $F(\Pi, i)$ is finite, suppose the execution time of $S|_i$ is $n \cdot p$, then $\forall d \in [0, p)$, just n time units of color C_d are assigned to $S|_i$.*

Proof. Let increasing sequence M contain the $n \cdot p$ time units assigned to $S|_i$. We first prove that $\forall j \in [0, n \cdot p)$, $(M|_{j+1} - M|_j) \mod p = 1$. Let $t_1 = M|_j$ and $t_2 = M|_{j+1}$. It is obvious true when $t_2 - t_1 = 1$; otherwise, $S|_i$ is interrupted at time $t_1 + 1$ and resumed at time t_2. Since Γ is P-stable, each job executed between $t_1 + 1$ and t_2

must be released after t_1 and finished no later than t_2. And since S is p-harmonic and there is no idle time unit between $t_1 + 1$ and t_2, $(t_2 - t_1 - 1) \mod p = 0$. It follows $(t_2 - t_1) \mod p = 1$. Therefore, any p consecutive time units in M have p different colors. The property follows immediately. □

Lemma 5.3.2 *Given two regular job systems* $\Pi = (\mathcal{A}^{(1)}, S, \Gamma, \prec_t)$ *and* $\Pi' = (\mathcal{A}^{(1)}, S, \Gamma^{(p,d)}, \prec_t)$ *where* S *is* p*-harmonic and* Γ *is P-stable,* $\forall k \geq 0$*, suppose time unit* $(k \cdot p + d)$ *in* Π *is assigned to job* J *(resp. idle), then time frame* k *in* Π' *is also assigned to* J *(resp. idle).*

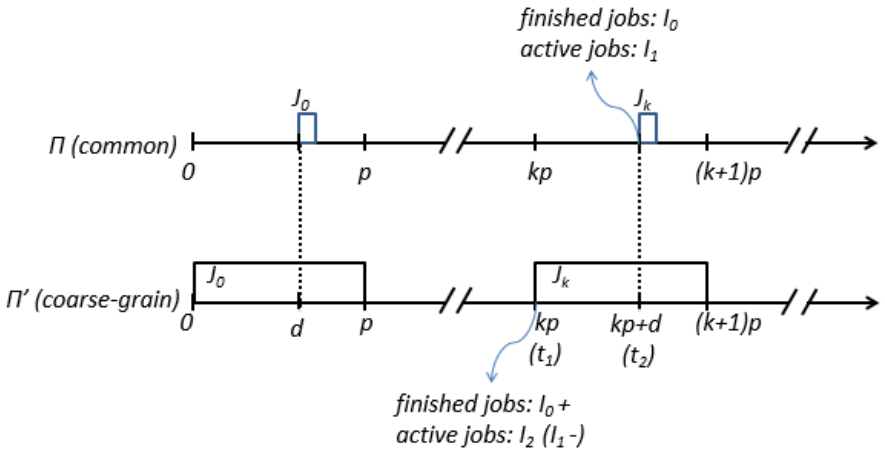


Figure 5.5: Matching a Common Single-Resource Schedule with a Coarse-grain Schedule

Proof. Since Π and Π' have the same priority order and tie-break order on S, they have the same scheduling order on S. Show the property by induction on k.

Check the base case, $k = 0$. If time unit d in Π is idle, there is no job released in time interval $[0, d]$ such that time frame 0 in Π' is idle. Otherwise, if time unit d

in Π is assigned to a job J_0, J_0 has the highest scheduling order among the jobs released in $[0, d]$ such that time frame 0 in Π' is assigned to J_0.

Suppose the property is true till $(k-1) \geq 0$. Let I_0 denote the set of jobs that have finished at time $t_2 = (k \cdot p + d)$ in Π. $\forall J' \in I_0$, suppose the execution time of J' is $n' \cdot p$, then by Lemma 5.3.1, J' has been assigned n' time units of color C_d. By the induction assumption, J' has already been assigned n' time frames before time $t_1 = k \cdot p$ in Π'. Therefore, all the jobs in I_0 have finished at time t_1 in Π'.

case 1: If time unit t_2 in Π is idle, then I_0 includes all jobs released in $[0, t_2]$. Since all jobs in I_0 have finished at time t_1 in Π', time frame k is idle.

case 2: As shown in Figure 5.5, time unit t_2 in Π is assigned to a job J_k. Let I_1 denote the set of active jobs at time t_2 in Π, then J_k has the highest scheduling order in I_1. Suppose the execution time of J_k is $n \cdot p$, then by Lemma 5.3.1, J_k has been assigned at most $(n-1)$ time units of color C_d at time t_1 in Π. By the induction assumption, J_k has been assigned at most $(n-1)$ time frames at time t_1 in Π'. It follows that J_k is still active at time t_1 in Π'. Let I_2 denote the set of the active jobs[†] at time t_1 in Π', then $J_k \in I_2$. Since all the jobs in I_0 have finished at time t_1 in Π', $I_2 \subseteq I_1$. Since J_k has the highest scheduling order in I_1, J_k also has the highest scheduling order in I_2. It follows that time frame k in Π' is assigned to J_k. □

Lemma 5.3.3 shows a property on job finish times in the coarse-grain scenario. We will use it to check the feasibility of job systems on regular partitions to fulfill

[†]Due to the early-release mechanism of the coarse-grain scheduling, I_2 also includes the jobs released in $(t_1, t_2]$.

our transformation method.

Lemma 5.3.3 *Suppose Ω is a sequence containing p regular job systems and $\Omega|_d = (\mathcal{A}^{(1)}, S, \Gamma^{(p,d)}, \prec_t)$ for $d = 0, 1, ..., p-1$, where S is p-harmonic and Γ is P-stable. Let $\Pi = (\mathcal{A}^{(1)}, S, \Gamma, \prec_t)$, then $\forall i \in [0, |S|)$,*

(1) if $F(\Pi, i)$ is finite, $\forall d \in [0, p), F(\Omega|_d, i) \leq F(\Pi, i) + p - d - 1$ (the equality holds iff $d = (F(\Pi, i) - 1) \mod p$);

(2) if $F(\Pi, i)$ is infinite, $\exists d \in [0, p)$, $F(\Omega|_d, i)$ is infinite.

Proof. Let $J = S|_i$.

part (1): Suppose t_1 is the last time unit of color C_d assigned to J in Π, then by Lemma 5.3.2, $F(\Omega|_d, i) = t_1 + p - d$. Also, since $t_1 \leq F(\Pi, i) - 1$, the property is true. The equality only holds when $t_1 = F(\Pi, i) - 1 \Leftrightarrow d = (F(\Pi, i) - 1) \mod p$.

part (2): Suppose t_2 is the last time unit assigned to J in Π. Let $d = (t_2 + 1) \mod p$. Since $F(\Pi, i)$ is infinite, suppose the execution time of J is $n \cdot p$, then there are at most $n - 1$ time units of color C_d assigned to J in Π. It follows that $F(\Omega|_d, i)$ is infinite because J is assigned at most $(n - 1)$ time frames in $\Omega|_d$. □

5.4 Transparent Job Scheduling via a Transformation Method

In this section, we completely present how to achieve transparent job scheduling on regular partitions. As mentioned, our strategy is to transform a scheduling

problem on a regular partition to one on a single resource.

5.4.1 Job Scaling

Job scaling (Def.s 5.4.1 and 5.4.2) is a part of the transformation method.

Definition 5.4.1 *$J^{(p,q)}$ denotes a job as the result of a scale operation on a job J with a tuple (p, q), where $J^{(p,q)} = (p \cdot C(J), q \cdot R(J), q \cdot D(J))$.*

Definition 5.4.2 *$S^{(p,q)}$ denotes a job sequence as the result of a scale operation on a job sequence S with a tuple (p, q), where $|S^{(p,q)}| = |S|$, and $\forall i \in [0, |S|), S^{(p,q)}|_i = (S|_i)^{(p,q)}$.*

Other than "P-stable", a further restriction "P-scalable" is needed on scheduling polices for the transformation. As shown in Def. 5.3, we require that a job sequence's priority order for such a policy to remain unchanged after a scale operation is applied.

Policy	Job Priority	P-Stable	P-Scalable
RR	N.A.	No	No
LLF	N.A.	No	No
DM	$D(J) - R(J)$	Yes	Yes
EDF	$D(J)$	Yes	Yes
FIFO	$R(J)$	Yes	Yes
N.A.	$R(J)/D(J)$	Yes	Yes
N.A.	$R(J) \cdot R(J) + D(J)$	Yes	No
N.A.	$D(J) - R(J) - C(J)$	Yes	No

Table 5.2: Policy and Job Priority

Definition 5.4.3 *A scheduling policy is* ***P-scalable*** *iff it is P-stable, and* $\forall J_1, J_2$ *and* (p, q)*, the priority order between* $J_1^{(p,q)}$ *and* $J_2^{(p,q)}$ *is the same as the one between* J_1 *and* J_2*.*

Based on Def. 5.4.1, we can easily conclude that a scheduling policy is P-scalable if the job priority is determined by a linear function on release time and deadline, $a \cdot R(J) + b \cdot D(J)$, where a, b are constants. In Table 5.2, we show whether some popular policies have the P-stable and P-scalable properties, including RR (Round Robin), LLF (Least Laxity First), DM, EDF and FIFO (First In First Out). Notice that EDF and DM are both P-scalable, so our transformation method applies to them.

5.4.2 Connect Coarse-grain Scheduling to Regular-Partition Scheduling

Now, we start to consider the scheduling problems on regular partitions. For convenience, we still use "time unit i" to denote the i-th time unit of a regular-partition assignment. Notice that time unit i of a regular-partition assignment does not necessarily start at time i because unlike a dedicated resource, the time units on a regular-partition assignment are not continuous everywhere.

Lemma 5.4.1 performs Step ② in Figure 5.3, which matches a scheduling problem on a regular-partition assignment to a coarse-grain scheduling problem. To show the intuition, we briefly explain a simple case of the matching in Figure 5.6 (the case $q = 1$ in Lemma 5.4.1), where Γ is P-scalable, $\Pi' = (\mathcal{A}_d^{(p,1)}, S, \Gamma, \prec_t)$ describes a scheduling

problem on a regular-partition assignment and $\Pi = (\mathcal{A}^{(1)}, S^{(p,1)}, \Gamma^{(p,d)}, \prec_t)$ describes a coarse-grain scheduling problem. Since Π' and Π have the same priority order and tie-break order on S and $S^{(p,1)}$ respectively, they have the same scheduling order on S and $S^{(p,1)}$, respectively. If the first time frame in Π is assigned to $J_0^{(p,1)}$, then $J_0^{(p,1)}$ has the highest scheduling priority at time d in Π (early-release mechanism in the coarse-grain scheduling). Therefore, J_0 has the highest scheduling priority at time d in Π'. It follows that the first time unit in Π' is assigned to J_0. Similarly, the assignments of the second, third, ... time frames/units are also matched.

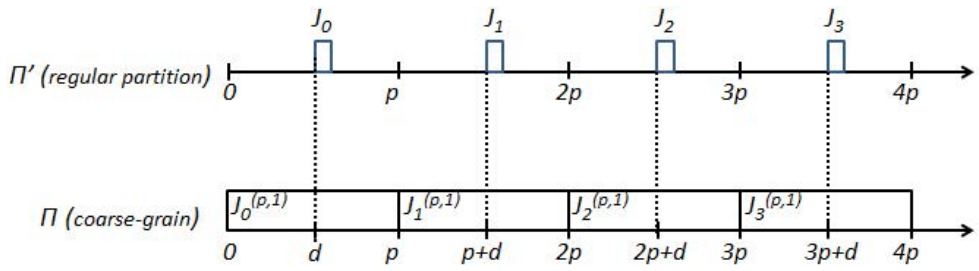


Figure 5.6: A Simple Case of Matching a Regular-Partition Schedule with a Coarse-grain Schedule

Lemma 5.4.1 *Given two regular job systems* $\Pi = (\mathcal{A}^{(1)}, S^{(p,q)}, \Gamma^{(p,d)}, \prec_t)$ *and* $\Pi' = (\mathcal{A}_d^{(p,q)}, S, \Gamma, \prec_t)$ *where* Γ *is P-scalable,* $\forall k \geq 0$*, time unit* k *in* Π' *is assigned to* $S|_i$ *(resp. idle) if time frame* k *in* Π *is assigned to job* $S^{(p,q)}|_i$ *(resp. idle).*

Proof. Show the property by induction on k.

Check the base case, $k = 0$.

case 1: If time frame 0 in Π is idle, then no job is released in $[0, d]$ in Π. It follows that no job is released in $[0, \lfloor \frac{d}{q} \rfloor]$ in Π'. By Lemma 2.2, time unit 0 in Π' starts at

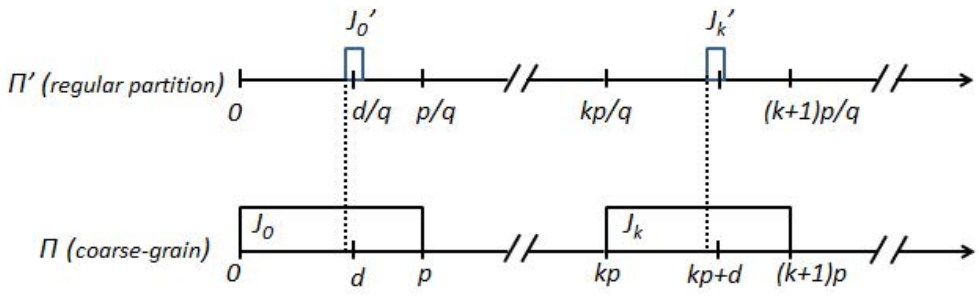


Figure 5.7: The General Case of Matching a Regular-Partition Schedule with a Coarse-grain Schedule

time $\lfloor \frac{d}{q} \rfloor$. This time unit is idle because no job is released before it in Π'.

case 2: As shown in Figure 5.7, if time frame 0 in Π is assigned to job $J_0 = S^{(p,q)}|_{i_0}$, then J_0 has the highest scheduling priority among jobs released in $[0, d]$ in Π. Meanwhile, no job is released in $(q \cdot \lfloor \frac{d}{q} \rfloor, d]$ because the release time of each job in $S^{(p,q)}$ is a multiple of q (Def.s 5.4.1 and 5.4.2). It follows that J_0 has the highest scheduling priority among jobs released in $[0, q \cdot \lfloor \frac{d}{q} \rfloor]$ in Π. Since Γ is P-scalable and Π' has the same tie-break order with Π, it follows that job $J'_0 = S|_{i_0}$ has the highest scheduling priority among jobs released in $[0, \lfloor \frac{d}{q} \rfloor]$ in Π'. It follows that time unit 0 in Π' is assigned to J'_0.

Suppose the property is true till $k - 1 \geq 0$. Due to the synchronized time frame/unit assignments in the first k time frames, if a job $J^{(p,q)}$ is finished (resp. unfinished) before time $k \cdot p$ in Π, its corresponding job J is also finished (resp. unfinished) before time $\frac{k \cdot p}{q}$ in Π'. It follows that the active job sets at time $k \cdot p$ in Π and Π' are synchronized. As in the base case, the property is true similarly. □

Similar to Lemma 5.3.3, the following lemma shows a property on job finish times.

Lemma 5.4.2 *Given two regular job systems* $\Pi = (\mathcal{A}^{(1)}, S^{(p,q)}, \Gamma^{(p,d)}, \prec_t)$ *and* $\Pi' = (\mathcal{A}_d^{(p,q)}, S, \Gamma, \prec_t)$ *where* Γ *is P-scalable,* $\forall i \in [0, |S|)$,

(1) if $F(\Pi, i)$ *is finite,* $F(\Pi, i) - q \cdot F(\Pi', i) \in [p - d - q, p - d)$.

(2) if $F(\Pi, i)$ *is infinite,* $F(\Pi', i)$ *is also infinite.*

Proof. part (1): Suppose $S^{(p,q)}|_i$ finishes at time frame k in Π, then by Lemma 5.4.1, $S|_i$ finishes at time unit k in Π'. Let time unit k in Π' starts at time t. From Figure 5.7, $\frac{k \cdot p + d}{q} \in [t, t+1) \Rightarrow t \in (\frac{k \cdot p + d}{q} - 1, \frac{k \cdot p + d}{q}]$, then $F(\Pi, i) - q \cdot F(\Pi', i) = (k+1) \cdot p - (t+1) \cdot q \in \Big[(k+1) \cdot p - (\frac{k \cdot p + d}{q} + 1) \cdot q, (k+1) \cdot p - \frac{k \cdot p + d}{q} \cdot q\Big)$. The property follows.

part (2): Immediately follows from Lemma 5.4.1. □

5.4.3 Equivalence between Regular-Partition Scheduling and Single-Resource Scheduling

It is ready for Step ③ in Figure 5.3 by combining the results on job finish times in Lemmas 5.3.2 and 5.4.2. As the result, Theorem 5.4.1 thoroughly presents a transformation method – a scheduling problem on a regular partition can be transformed into one on a single resource that has the same schedulability.

Theorem 5.4.1 *Suppose a preemptive hard-real-time scheduling policy* Γ *is P-scalable, then* $(\mathcal{A}_d^{(p,q)}, S, \Gamma, \prec_t)$ *is feasible for any* $d \in [0, p)$ *iff* $(\mathcal{A}^{(1)}, S^{(p,q)}, \Gamma, \prec_t)$ *is feasible.*

Proof. Let $\Pi = (\mathcal{A}^{(1)}, S^{(p,q)}, \Gamma, \prec_t)$.

if part: $\forall d \in [0, p)$, let $\Pi' = (\mathcal{A}_d^{(p,q)}, S, \Gamma, \prec_t)$ and $\Pi'' = (\mathcal{A}^{(1)}, S^{(p,q)}, \Gamma^{(p,d)}, \prec_t)$. $\forall i \in [0, |S|)$, if Π is feasible, by Lemma 5.3.3,

$$F(\Pi'', i) \leq F(\Pi, i) + p - d - 1. \tag{i}$$

It follows that $F(\Pi'', i)$ is finite, then by Lemma 5.4.2,

$$F(\Pi'', i) - q \cdot F(\Pi', i) \geq p - d - q. \tag{ii}$$

(i) and (ii) conclude

$q \cdot F(\Pi', i) \leq F(\Pi, i) + q - 1$

$\Rightarrow q \cdot F(\Pi', i) \leq D(S^{(p,q)}|_i) + q - 1$ *Def. 5.2.7*

$\Rightarrow q \cdot F(\Pi', i) \leq q \cdot D(S|_i) + q - 1$ *Def. 5.4.1*

$\Rightarrow F(\Pi', i) \leq D(S|_i)$.

Therefore, Π' is feasible because it has no deadline miss.

only if part: Let Ω be a sequence containing p regular job systems where $\Omega|_d = (\mathcal{A}^{(1)}, S^{(p,q)}, \Gamma^{(p,d)}, \prec_t)$ for $d = 0, 1, ..., p-1$. If Π is infeasible, $\exists i \in [0, |S|)$,

$$F(\Pi, i) > q \cdot D(S|_i). \tag{iii}$$

case 1: $F(\Pi, i) = \infty$. By Lemma 5.3.3, $\exists d \in [0, p)$, $F(\Omega|_d, i) = \infty$. Let $\Pi' = (\mathcal{A}_d^{(p,q)}, S, \Gamma, \prec_t)$. By Lemma 5.4.2, $F(\Pi', i) = \infty$. It follows that Π' is infeasible.

case 2: $F(\Pi, i)$ is finite. By Lemma 5.3.3, $\exists d \in [0, p)$,

$$F(\Omega|_d, i) = F(\Pi, i) + p - d - 1. \tag{iv}$$

Let $\Pi' = (\mathcal{A}_d^{(p,q)}, S, \Gamma, \prec_t)$. By Lemma 5.4.2,

$$F(\Omega|_d, i) - q \cdot F(\Pi', i) \leq p - d - 1. \quad \text{(v)}$$

(iv) and (v) conclude

$$q \cdot F(\Pi', i) \geq F(\Pi, i)$$

$$\Rightarrow F(\Pi', i) > D(S|_i) \qquad \textit{Inequation (iii)}$$

It follows that Π' is infeasible. □

Scheduling Example: Schedule $S = \langle (1, 0, 2), (2, 1, 4) \rangle$ by EDF on a regular partition of weight $\frac{2}{3}$. From Theorem 5.4.1, we just need to check the schedulability of $S^{(3,2)} = \langle (3, 0, 4), (6, 2, 8) \rangle$ by EDF on a single resource. Since $S^{(3,2)}$ is unschedulable by EDF on a single resource, S is unschedulable by EDF on a regular partition of weight $\frac{2}{3}$.

5.5 Applying the Transformation Method to periodic tasks

Till now, we focus on job systems where each job has only one single request. However, tasks with multiple requests are most common in actual real-time systems. To illustrate the applicability of our method, we shall present how to apply it to scheduling periodic tasks whose requests arrive periodically. We first define periodic tasks as follows.

Definition 5.5.1 *$\tau = (c, p, d, o)$ denotes a periodic task, where c, p, d, and o are its execution time, period, deadline, and offset, respectively. For convenience, we*

use $C(\tau)$, $P(\tau)$, $D(\tau)$ and $O(\tau)$ to denote τ's execution time, period, deadline, and offset, respectively.

5.5.1 Regular Periodic Task System

To describe a scheduling problem for periodic tasks, we define a regular periodic task system as follows. Notice that it also has a tie-break order to remove ties.

Definition 5.5.2 *$\Psi = (R, \mathcal{T}, \Gamma, \prec_T)$ denotes a **regular periodic task system**, where R is a regular-partition assignment; $\mathcal{T}$ is a sequence of independent periodic tasks; Γ is a preemptive hard-real-time scheduling policy; and $\prec_T$ is a strict total order on $[0, |\mathcal{T}|)$ as the tie-break order.*

The key to our solution for periodic tasks is to convert a periodic task sequence to a job sequence as follows. A periodic task is naturally able to generate an infinite series of jobs as in Def. 5.5.3. For a periodic task sequence, we just need to add a rule to sort the generated jobs as in Def. 5.5.4.

Definition 5.5.3 *Given a periodic task τ, $release(\tau, k) = O(\tau) + k \cdot P(\tau)$ and $deadline(\tau, k) = release(\tau, k) + D(\tau)$ denote the release time and the deadline of the k-th ($k \geq 0$) job of τ, respectively.*

Definition 5.5.4 *For a given periodic task sequence $\mathcal{T}$, S is $\mathcal{T}$'s **congruent job sequence** iff $\forall k \geq 0$ and $i \in [0, |\mathcal{T}|)$, $S|_{k \cdot |\mathcal{T}|+i} = (C(\mathcal{T}|_i), release(\mathcal{T}|_i, k), deadline(\mathcal{T}|_i, k))$.*

Then we can convert a regular periodic task system to a corresponding job system as in Def. 5.5.5. The only concern is how to generate a tie-break order for the

generated job sequence. Suppose a tie happens between two jobs belonging to tasks τ_1 and τ_2, respectively. If τ_1 and τ_2 are different, we just keep the original tie-break order between τ_1 and τ_2; otherwise, the earlier job wins the tie.

Definition 5.5.5 *For a given regular periodic task system $\Psi = (R, \mathcal{T}, \Gamma, \prec_T)$, regular job system $\Pi = (R, S, \Gamma, \prec_t)$ is Ψ's **congruent job system** iff S is $\mathcal{T}$'s congruent job sequence and when a tie happens between $S|_{k_1 \cdot |\mathcal{T}| + i_1}$ and $S|_{k_2 \cdot |\mathcal{T}| + i_2}$, $(k_1 \cdot |\mathcal{T}| + i_1) \prec_t (k_2 \cdot |\mathcal{T}| + i_2)$ iff $i_1 \prec_T i_2$ OR $i_1 = i_2$ and $k_1 < k_2$.*

Similarly, a regular periodic task system is regarded as feasible if it has no deadline miss.

Definition 5.5.6 *A regular periodic task system is feasible iff all jobs generated by its tasks meet their deadlines.*

The jobs generated in a regular periodic task system Ψ exactly match the jobs in its congruent job system Π (Def. 5.5.4). Meanwhile, Ψ and Π have the same scheduling policy and their tie-break orders are also exactly matching (Def. 5.5.5). Corollary 5.5.1 immediately follows.

Corollary 5.5.1 *A regular periodic task system is feasible iff its congruent job system is feasible.*

We also need to scale periodic tasks when applying the transformation method.

Definition 5.5.7 *$\tau^{(p,q)}$ denotes a periodic task as the result of a scale operation on a periodic task τ with a tuple (p, q), where $\tau^{(p,q)} = (p \cdot C(\tau), q \cdot P(\tau), q \cdot D(\tau), q \cdot O(\tau))$.*

Definition 5.5.8 *$\mathcal{T}^{(p,q)}$ denotes the a periodic task sequence as the result of a scale*

operation on a periodic task sequence $\mathcal{T}$ *with a tuple* (p, q)*, where* $|\mathcal{T}^{(p,q)}| = |\mathcal{T}|$*, and* $\forall i \in [0, |\mathcal{T}|)$*,* $\mathcal{T}^{(p,q)}|_i = (\mathcal{T}|_i)^{(p,q)}$*.*

Finally, we can apply the transformation method to a periodic task sequence as follows.

Theorem 5.5.1 *Given a P-scalable policy* Γ *and a periodic task sequence* $\mathcal{T}$*,* $(\mathcal{A}_d^{(p,q)}, \mathcal{T}, \Gamma, \prec_T)$ *is feasible for any* $d \in [0, p)$ *iff* $(\mathcal{A}^{(1)}, \mathcal{T}^{(p,q)}, \Gamma, \prec_T)$ *is feasible.*

Proof. Immediately follows from Corollary 5.5.1 and Theorem 5.4.1. □

5.5.2 Earliest Deadline First

EDF is one of the most popular scheduling policies. Liu and Layland [8] present its utilization bound for synchronous[‡] periodic tasks with implicit deadlines[§] on a single resource, and Labetoulle [18] extends it to the asynchronous case as in Lemma 5.5.1.

Definition 5.5.9 *The Utilization of a periodic task sequence* $\mathcal{T}$ *is denoted by* $U(\mathcal{T}) = \sum_{i=0}^{|\mathcal{T}|-1} \frac{C(\mathcal{T}|_i)}{P(\mathcal{T}|_i)}$*.*

Lemma 5.5.1 [18] *Given a periodic task sequence* $\mathcal{T}$ *where* $D(\mathcal{T}|_i) = P(\mathcal{T}|_i)$ *for any* $i \in [0, |\mathcal{T}|)$*,* $(\mathcal{A}^{(1)}, \mathcal{T},$ *EDF*$, \prec_T)$ *is feasible iff* $U(\mathcal{T}) \leq 1$*.*

By our transformation method presented in Theorem 5.5.1, we can easily obtain the utilization bound of EDF for periodic tasks with implicit deadlines on a regular

[‡]The offset of each task is 0.

[§]The deadline of each task coincides with its period.

partition as in Theorem 5.5.2. Notice that Shigero et al. [12] only obtain the corresponding result for the synchronous case, and our transformation method solves the asynchronous case.

Theorem 5.5.2 *Given a periodic task sequence $\mathcal{T}$ where $D(\mathcal{T}|_i) = P(\mathcal{T}|_i)$ for any $i \in [0, |\mathcal{T}|)$, $(\mathcal{A}_d^{(p,q)}, \mathcal{T}, EDF, \prec_T)$ is feasible for any $d \in [0, p)$ iff $U(\mathcal{T}) \leq \frac{q}{p}$.*

Proof. Immediately follows from Lemma 5.5.1 and Theorem 5.5.1. □

For an asynchronous periodic task set with arbitrary deadlines on a single resource, Leung and Merril [15], and Baruah et al. [16] show its feasibility interval as in Lemma 5.5.2. We can directly use it to check the same case on a regular partition with Theorem 5.5.1. Furthermore, we can also grab any new schedulability test for EDF, such as [17].

Lemma 5.5.2 [15, 16] *Given a periodic task sequence $\mathcal{T}$, $(\mathcal{A}^{(1)}, \mathcal{T}, EDF, \prec_T)$ is feasible iff $U(\mathcal{T}) \leq 1$ and no deadline misses in time interval $[0, O^{max} + 2H)$, where $O^{max} = \max\{O(\mathcal{T}|_i) : i \in [0, |\mathcal{T}|)\}$ and $H = \text{lcm}\{P(\mathcal{T}|_i) : i \in [0, |\mathcal{T}|)\}$.*

5.5.3 Deadline Monotonic

Similarly, we can obtain the utilization bound of DM for a synchronous periodic task set with implicit deadlines on a regular partition as in Theorem 5.5.3 based on the corresponding single-resource result in [8]. We can also utilize a complex sufficient and necessary schedulability test of DM for such a task set presented in [19].

Lemma 5.5.3 [8] *Given a periodic task sequence $\mathcal{T}$ with n tasks where $D(\mathcal{T}|_i) = P(\mathcal{T}|_i)$ and $O(\mathcal{T}|_i) = 0$ for any $i \in [0, n)$, $(\mathcal{A}^{(1)}, \mathcal{T}, DM, \prec_T)$ is feasible if $U(\mathcal{T}) < n(\sqrt[n]{2} - 1)$.*

Theorem 5.5.3 *Given a periodic task sequence $\mathcal{T}$ with n tasks where $D(\mathcal{T}|_i) = P(\mathcal{T}|_i)$ and $O(\mathcal{T}|_i) = 0$ for any $i \in [0, n)$, $(\mathcal{A}_d^{(p,q)}, \mathcal{T}, DM, \prec_T)$ is feasible for any $d \in [0, p)$ if $U(\mathcal{T}) < n(\sqrt[n]{2} - 1) \cdot \frac{q}{p}$.*

Proof. Immediately follows from Lemma 5.5.3 and Theorem 5.5.1. □

For more general cases, we can directly use the results on feasible intervals as follows.

Lemma 5.5.4 [20] *Given a periodic task sequence $\mathcal{T}$ where $D(\mathcal{T}|_i) < P(\mathcal{T}|_i)$ and $O(\mathcal{T}|_i) = 0$ for any $i \in [0, |\mathcal{T}|)$, $(\mathcal{A}^{(1)}, \mathcal{T}, DM, \prec_T)$ is feasible iff no deadline misses in time interval $[0, D^{max}]$, where $D^{max} = \max\{D(\mathcal{T}|_i) : i \in [0, |\mathcal{T}|)\}$.*

Lemma 5.5.5 [20] *Given a periodic task sequence $\mathcal{T}$, $(\mathcal{A}^{(1)}, \mathcal{T}, DM, \prec_T)$ is feasible iff $U(\mathcal{T}) \leq 1$ and no deadline misses in time interval $[0, O^{max} + 2H)$, where $O^{max} = \max\{O(\mathcal{T}|_i) : i \in [0, |\mathcal{T}|)\}$ and $H = \text{lcm}\{P(\mathcal{T}|_i) : i \in [0, |\mathcal{T}|)\}$.*

5.6 Contribution and Future Work

We present a transformation method to achieve transparent task scheduling in a 2-layer hierarchical real-time system based on the RRP Model. With this method, we can reuse off-the-shelf single-resource scheduling techniques to solve task-scheduling

problems on regular partitions, avoiding the development of new task-scheduling techniques. Furthermore, our work greatly expands the study field of task scheduling on temporal resource partitions because current research focuses on a scheduling policy (DM or EDF) and a task model (limited periodic or sporadic) at a time. On the one hand, our transformation method applies to any preemptive hard-real-time policy which is P-scalable, such as DM, EDF, and FIFO. On the other hand, it deals with any task set containing periodic, sporadic, and aperiodic tasks.

In the future, we can extend our transformation method in the following possible aspects:

• Soft-real-time policies – our method could be easily applied to soft-real-time policies because it is based on the finish-time analysis of jobs.

• Dynamic job-priority policies – we may need to adapt our method to deal with policies where a job's priority is not a constant number, such as RR and LLF.

• Non-preemptive policies – our method does not seem to work with non-preemptive policies. New methods are required for them.

Chapter 6

Conclusion

The Virtualization Technology allows people to integrate multiple operating systems and applications onto one physical platform. In the real-time area, it encounters the problem that the original real-time scheduling policies cannot work in the new virtualized system without change. Our work has solved this problem by extending the current Regularity-based Resource (RRP) Model. The RRP Model was announced to be the most transparent Hierarchical Real-Time Resource Model and had the potential to build up a practical real-time Virtual Machine Monitor (VMM). However, as shown in Table 1.2, there has been merely limited and primitive results at both the resource level and the task level in this model. At the resource level, we succeed to solve the resource-partitioning problem for both uniresources and multiresources. In the multiresource scenario, we introduce novel algorithms for resource partitioning with both global and partitioned strategies. Our experimental results show that

they improve the overall resource utilization and scheduling performance, significantly. At the task level, we study task scheduling on regular partitions to show that they have the transparency property in comprehensive cases. A scheduling problem of a regular partition can always be transformed into an equivalent problem of a dedicated single resource. Therefore, we do not need to develop new schedulability test and utilization bound in the virtualized real-time system. With these theoretical fundamentals for resource management and scheduling in the real-time VMM, researchers and developers could focus on other problems, such as fault tolerance and task/resource dependency.

Bibliography

[1] A. K. Mok and X. Feng. Towards compositionality in real-time resource partitioning based on regularity bounds. *Proceedings of the Real-Time Systems Symposium (RTSS)*, Pages: 129 - 138, 2001.

[2] A. K. Mok, X. Feng, and D. Chen. Resource partition for real-time systems. *Proceedings of the Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Pages: 75 - 84, 2001.

[3] X. Feng. Design of real-time virtual resource architecture for largescale embedded systems. *Ph.D. Dissertation*, The University of Texas at Austin, 2004.

[4] Y. Li, A. M. K. Cheng, A. K. Mok. Regularity-based partitioning of uniform resources in real-time systems. *Proceedings of the International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Pages: 368 - 377, 2012.

[5] S. Baruah, N. Cohen, G. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, Volume: 15, Pages: 600 - 625, 1996.

[6] S. Baruah, J. Gehrke, and G. Plaxton. Fast scheduling of periodic tasks on multiple resources. *Proceedings of the International Parallel Processing Symposium* (IPPS), Pages: 280 - 288, 1995.

[7] S. Shigero, M. Takashi, and H. Kei. On the schedulability conditions on partial time slots. *Proceedings of the International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Pages: 166 - 173, 1999.

[8] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming environment in a hard real-time environment. *Journal of the ACM (JACM)*, Volume: 20, Issue: 1, Pages: 46 - 61, 1973.

[9] B. Andersson and K. Bletsas. Sporadic multiprocessor scheduling with few preemptions. *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, Pages: 243 - 252, 2008.

[10] S. Kato and N. Yamasaki. Semi-partitioned fixed-priority scheduling on multiprocessors. *Proceedings of the Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Pages: 23 - 32, 2009.

[11] I. Shin, A. Easwaran, and I. Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, Pages: 181 - 190, 2008.

[12] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. *Proceedings of the Real-Time Systems Symposium (RTSS)*, Pages: 2 - 13, 2003.

[13] Y. Li, and A. M. K. Cheng. Static approximation algorithms for regularity-based resource partitioning. *Proceedings of the Real-Time Systems Symposium (RTSS)*, Pages: 137 - 148, 2012.

[14] A. Easwaran, M. Anand, and I. Lee. Compositional analysis framework using EDP resource models. *Proceedings of the Real-Time Systems Symposium (RTSS)*, Pages: 129 - 138, 2007.

[15] J.-T. Leung and M. Merril. A note on preemptive scheduling of periodic real-time tasks. *Information Processing Letters*, Volume: 11, Issue: 3, Pages: 115 - 118, 1980.

[16] S. Baruah, L. Rosier, and R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor. *The Journal of Real-Time Systems*, Volume: 2, Issue: 4, Pages: 301 - 324, 1990.

[17] R. Pellizzoni and G. Lipari. A new sufficient feasibility test for asynchronous real-time periodic task sets. *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, Pages: 204 - 211, 2004.

[18] J. Labetoulle. Some theorems on real-time scheduling. *E. Gelenbe and R. Mahl (eds.), Computer Architecture and Networks*, Amsterdam: North-Holland, Pages: 285 - 293, 1974.

[19] J. P. Lehoczky, L. Sha, and Y. Ding. The rate-monotonic scheduling algorithm: exact characterization and average case behaviour. *Proceedings of the Real-Time Systems Symposium (RTSS)*, Pages: 166 - 171, 1989.

[20] J. Goossens. Scheduling of hard real-time periodic systems with various kinds of deadline and offset constraints. *Ph.D. Dissertation*, University of Brussels, 1999.

[21] K. Jeffay, D. F. Stanat, and C. U. Martel. On non-preemptive of period and sporadic tasks. *Proceedings of the Real-Time Systems Symposium (RTSS)*, Pages: 129 - 139, 1991.

[22] M. G. Harbour, M. H. Klein, J. P. Lehoczky. Timing analysis for fixed-priority scheduling of hard real-time systems. *IEEE Transactions on Software Engineering*, Volume: 20, Issue: 1, Pages: 13 - 28, 1994.

[23] R. I. Davis and A. Burns. Response time upper bounds for fixed priority real-time systems. *Proceedings of the Real-Time Systems Symposium (RTSS)*, Pages: 407 - 418, 2008.

[24] S. Lauzac, R. Melhem, and D. Mosse. An improved rate-monotonic admission control and its applications. *IEEE Transactions on Computers*, Volume: 52, Issue: 3, Pages: 337 - 350, 2003.

[25] E. Bini, G. C. Buttazzo, and G. M. Buttazzo. Rate monotonic analysis: the hyperbolic bound. *IEEE Transactions on Computers*, Volume: 52, Issue: 7, Pages: 933 - 942, 2003.

[26] R. I. Davis, A. Zabos, and A. Burns. Efficient exact schedulability tests for fixed priority real-time systems. *IEEE Transactions on Computers*, Volume: 57, Issue: 9, Pages: 1261 - 1276, 2008.

[27] Q. Lu, A. M. K. Cheng, and R. Davis. Tunable response time upper bound for fixed-priority real-time systems. *Proceedings of the Real-Time and Embedded Technology and Applications Symposium (RTAS) WIP*, Pages: 23 - 24, 2015.

[28] J. Ras and A. M. K. Cheng. Response time analysis for the abort-and-restart event handlers of the Priority-based Functional Reactive Programming (P-FRP) paradigm. *Proceedings of the Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Pages: 305 - 314, 2009.

[29] C. Belwal and A. M. K. Cheng. Determining actual response time in P-FRP. *Proceedings of the International Symposium on Practical Aspects of Declarative Languages (PADL)*, Pages: 250 - 264, 2011.

[30] J. Lin, W. Song, and A. M. K. Cheng. RealEnergy: a new framework and a case study to evaluate power-aware real-time scheduling algorithms. *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, Pages: 153 - 158, 2010.

[31] J. Lin, A. M. K. Cheng, D. Steel, and M. Y.-C. Wu, Scheduling mixed-criticality real-time tasks with fault tolerance. *Workshop on Mixed Criticality, in conjuction with IEEE RTSS*, Article No. 7, 2014.

[32] J.M. Lopez, J.L. Diaz, and D.F. Garcia. Utilization bound for EDF scheduling on real-time multiprocessor systems. *The Journal of Real-Time Systems*, Volume: 28, Issue: 1, Pages: 39 - 68, 2004.

[33] A. Bastoni, B.B. Brandenburg and J.H. Anderson. An empirical comparison of global, partitioned, and clustered multiprocessor EDF schedulers. *Proceedings of the Real-Time Systems Symposium (RTSS)*, Pages: 14 - 24, 2010.

[34] A. Bastoni, B. Brandenburg, and J. Anderson. Is semi-partitioned scheduling practical? *Proceedings of the Euromicro Conference on Real-Time Systems (E-CRTS)*, Pages: 125 - 135, 2011.

[35] Y. Li and A. M. K. Cheng. Transparent real-time task scheduling on temporal resource partitions. *IEEE Transaction on Computers (TC)*, Volume: 65, Issue: 5, Pages: 1646 - 1655, 2016.

[36] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. *Proceedings of the Real-Time Systems Symposium (RTSS)*, Pages: 4 - 13, 1998.

[37] Virtual PC. http://www.microsoft.com/windows/virtual-pc/default.aspx/

[38] Virtual Box. http://www.virtualbox.org/

[39] Xen. http://www.xenproject.org/

[40] Vmware. http://www.vmware.com/

[41] KVM. http://www.linux-kvm.org/

[42] K. J. Duda and D. R. Cheriton. Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. *Proceedings of the Symposium on Operating Systems Principles (SOSP)*, Pages: 261 - 276, 1999.

[43] Credit Scheduler. http://wiki.xensource.com/xenwiki/CreditScheduler.

[44] I. M. Leslie, D. Mcauley, R. Black, T. Roscoe, P. T. Barham, D. Evers, R. Fairbairns, and E. Hyden. The design and implementation of an operating system to support distributed multimedia applications. *IEEE Journal of Selected Areas in Communications*, Volume: 14, Issue: 7, Pages: 1280 - 1297, 1996.

[45] M. Behnam, I. Shin, T. Nolte, and M. Nolin. SIRAP: A synchronization protocol for hierarchical resource sharing in real-time open systems. *Proceedings of the International Conference on Embedded Software (EMSOFT)*, Pages: 279 - 288, 2007.

[46] Z. Deng and J. Liu. Scheduling real-time applications in an open environment. *Proceedings of the Real-Time Systems Symposium (RTSS)*, Pages: 308 - 319, 1997.

[47] I. Shin and I. Lee. Compositional real-time scheduling framework. *Proceedings of the Real-Time Systems Symposium (RTSS)*, Pages: 57 - 67, 2004.

[48] E. Bini, G. Buttazzo, and M. Bertogna. The multi supply function abstraction for multiprocessors. *Proceedings of the Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Pages: 294 - 302, 2009.

[49] S. Groesbrink, L. Almeida, M. de Sousa, and S. M. Petters. Towards certifiable adaptive reservations for hypervisor-based virtualization. *Proceedings of the Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Pages: 13 - 24, 2014.

[50] H. Leontyev and J. H. Anderson. A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, Pages: 191 - 200, 2009.

[51] G. Lipari and E. Bini. A framework for hierarchical scheduling on multiprocessors: from application requirements to run-time allocation. *Proceedings of the Real-Time Systems Symposium (RTSS)*, Pages: 249 - 258, 2010.

[52] M. Xu, L. T. X. Phan, I. Lee, O. Sokolsky, S. Xi, C. Lu, and C. D. Gill. Cache-aware compositional analysis of real-time multicore virtualization platforms. *Proceedings of the Real-Time Systems Symposium (RTSS)*, Pages: 1 - 10, 2010.

[53] Y. Wang and K. Lin. The implementation of hierarchical schedulers in the RED-Linux scheduling framework. *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, Pages: 231 - 238, 2000.

[54] T. Aswathanarayana, D. Niehaus, V. Subramonian, and C. Gill. Design and performance of configurable endsystem scheduling mechanisms. *Proceedings of the Real Time and Embedded Technology and Applications Symposium (RTAS)*, Pages: 32 - 43, 2005.

[55] F. Bruns, S. Traboulsi, D. Szczesny, E. Gonzalez, Y. Xu, and A. Bilgic. An evaluation of microkernel-based virtualization for embedded real-time systems. *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, Pages: 57 - 65, 2010.

[56] T. Cucinotta, G. Anastasi, and L. Abeni. Respecting temporal constraints in virtualised services. *Proceedings of the International Conference on Computers, Software & Applications (COMPSAC)*, Pages: 73 - 78, 2009.

[57] S. Xi, J. Wilson , C. Lu and C. Gill. RT-Xen: towards real-time hypervisor scheduling in Xen. *Proceedings of the ACM International Conference on Embedded Software (EMSOFT)*, Pages: 39 - 48, 2011.

[58] T. Cucinotta, F. Checconi, G. Kousiouris, D. Kyriazis, T. Varvarigou, A. Mazzetti, Z. Zlatev, J. Papay, M. Boniface, S. Berger, et al. Virtualised e-learning with real-time guarantees on the irmos platform. *Proceedings of the International Conference on Service-Oriented Computing and Applications (SOCA)*, Pages: 1 - 8, 2010.

[59] B. Lin and P. A. Dinda. Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling. *Proceedings of the ACM/IEEE conference on Supercomputing*, Pages: 8 - 20, 2005.

[60] J. Regehr and J. Stankovic. HLS: A framework for composing soft real-time schedulers. *Proceedings of the Real-Time Systems Symposium (RTSS)*, Pages: 3 - 14, 2001.

[61] M. Lee, A. Krishnakumar, P. Krishnan, N. Singh, and S. Yajnik. Supporting soft real-time tasks in the Xen hypervisor. *ACM Sigplan Notices*, Volume: 45, Issue: 7, Pages: 97 - 108, 2010.

[62] W. Zhang, S. Rajasekaran, T. Wood, and M. Zhu. MIMP: deadline and interference aware scheduling of hadoop virtual machines. *Proceedings of the International Symposium on Cluster, Cloud and Grid Computing*, Pages: 394 - 403, 2014.

[63] S. Govindan, A. R. Nath, A. Das, B. Urgaonkar and A. Sivasubramaniam. Xen and Co.: communication-aware CPU scheduling for consolidated Xen-based hosting platforms. *Proceedings of the International Conference on Virtual Execution Environments (VEE)*, Pages: 126 - 136, 2007.

[64] S. Yoo, K.-H. Kwak, J.-H. Jo, and C. Yoo. Toward under-millisecond I/O latency in Xen-ARM. *The Second Asia-Pacific Workshop on Systems*, Article No. 14, 2011.

[65] J. Lee, S. Xi, S. Chen, L. T. X. Phan, C. Gill, I. Lee, C. Lu and O. Sokolsky. Realizing compositional scheduling through virtualization. *Proceedings of the Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Pages: 13 - 22, 2012.

Yu Li has contributed to the following publications:

1. **Yu Li**, Albert M. K. Cheng. Static approximation algorithms for regularity-based resource partitioning. *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, Pages: 137 - 148, 2012.

2. **Yu Li**, Albert M. K. Cheng. Transparent real-time task scheduling on temporal resource partitions. *IEEE Transaction on Computers (TC)*, Volume: 65, Issue: 5, Pages: 1646 - 1655, 2016.

3. **Yu Li**, Albert M. K. Cheng, Aloysius K. Mok. Regularity-based partitioning of uniform resources in real-time systems. *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Pages: 368 - 377, 2012.

4. XingLiang Zou, Albert M. K. Cheng, **Yu Li**, and Yu Jiang. A temporal partition-based Linux CPU scheduler. *Proceedings of the IEEE International Conference on Embedded Software and System (ICESS)*, Pages: 705 - 708, 2014.

5. **Yu Li**, Albert M. K. Cheng. Toward a practical regularity-based model. Submitted to journal publication.